Scheduling

An event planner has to juggle many workers completing different tasks, some of which must be completed before others can begin. For example, the banquet tables would need to be arranged in the room before the catering staff could begin setting out silverware. The event planner has to carefully schedule things so that everything gets done in a reasonable amount of time.

The problem of scheduling is fairly universal. Contractors need to schedule workers and subcontractors to build a house as quickly as possible. A magazine needs to schedule writers, editors, photographers, typesetters, and others so that an issue can be completed on time.

Getting Started

To begin thinking about scheduling, let us consider an auto shop that is converting a car from gas to electric. A number of steps are involved. A time estimate for each task is given.

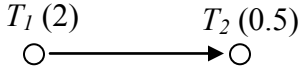
Task 1: Remove engine and gas parts (2 days)
Task 2: Steam clean the inside of the car (0.5 day)
Task 3: Buy an electric motor and speed controller (2 days for travel)
Task 4: Construct the part that connects the motor to the car's transmission (1 day)
Task 5: Construct battery racks (2 days)
Task 6: Install the motor (0.5 day)
Task 7: Install the speed controller (0.5 day)
Task 8: Install the battery racks (0.5 day)
Task 9: Wire the electricity (1 day)

Some tasks have to be completed before others – we certainly can't install the new motor before removing the old engine! There are some tasks, however, that can be worked on simultaneously by two different people, like constructing the battery racks and installing the motor.

To help us visualize the ordering of tasks, we will create a **digraph.**

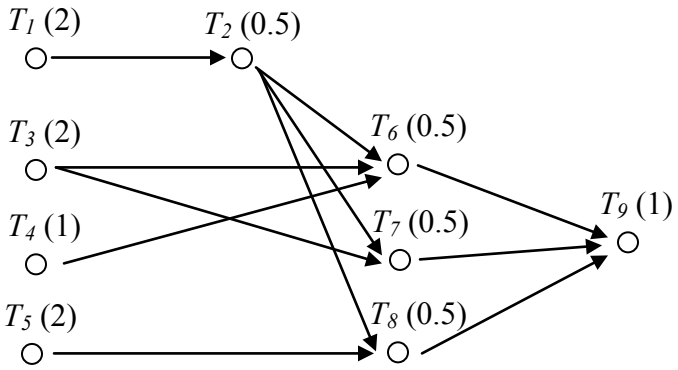
Digraph
A diagraph is a graphical representation of a set of tasks in which tasks are represented with dots, called vertices, and arrows between vertices are used to show ordering.

For example, this digraph shows that Task 1, notated T_1 for compactness, needs to be completed before Task 2. The number in parentheses after the task name is the time required for the task.



Example 1

A complete digraph for our car conversion would look like this:



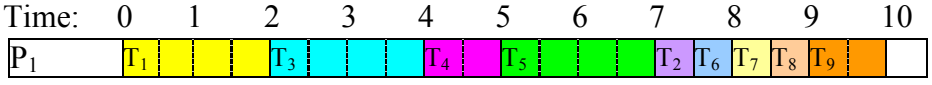
The time it takes to complete this job will partially depend upon how many people are working on the project.

> **Processors**
> In scheduling jargon, the workers completing the tasks are called **processors**. While in this example the processors are humans, in some situations the processors are computers, robots, or other machines.

For simplicity, we are going to make the very big assumptions that every processor can do every task, that they all would take the same time to complete it, and that only one processor can work on a task at a time.

> **Finishing Time**
> The **finishing** time is how long it will take to complete all the tasks. The finishing time will depend upon the number of processors and the specific schedule.

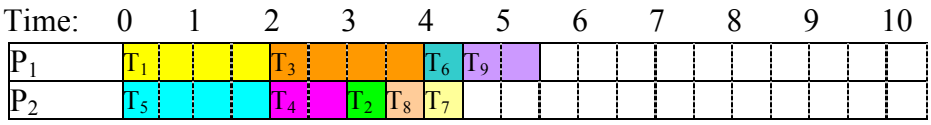
If we had only one processor working on this task, it is easy to determine the finishing time; just add up the individual times. We assume one person can't work on two tasks at the same time, ignore things like drying times during which someone could work on another task. Scheduling with one processor, a possible schedule would look like this, with a finishing time of 10 days.



In this schedule, all the ordering requirements are met. This is certainly not the only possible schedule for one processor, but no other schedule could complete the job in less time. Because of this, this is an **optimal schedule** with **optimal finishing time** – there is nothing better.

Optimal Schedule
An optimal schedule is the schedule with the shortest possible finishing time.

For two processors, things become more interesting. For small digraphs like this, we probably could fiddle around and guess-and-check a pretty good schedule. Here would be a possibility:

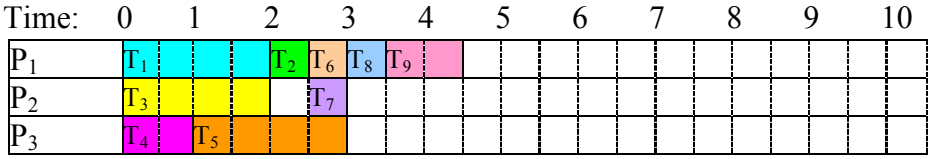


With two processors, the finishing time was reduced to 5.5 days. What was processor 2 doing during the last day? Nothing, because there were no tasks for the processor to do. This is called **idle time**.

Idle Time
Idle time is time in the schedule when there are no tasks available for the processor to work on, so they sit idle.

Is this schedule optimal? Could it have been completed in 5 days? Because every other task had to be completed before task 9 could start, there would be no way that both processors could be busy during task 9, so it is not possible to create a shorter schedule.

So how long will it take if we use three processors? About 10/3 = 3.33 days? Again we will guess-and-check a schedule:



With three processors, the job still took 4.5 days. It is a little harder to tell whether this schedule is optimal. However, it might be helpful to notice that since Task 1, 2, 6, and 9 have to be completed sequentially, there is no way that this job could be completed in less than 2+0.5+0.5+1 = 4 days, regardless of the number of processors. Four days is, for this digraph, the absolute minimum time to complete the job, called the **critical time**.

Critical Time
The critical time is the absolute minimum time to complete the job, regardless of the number of processors working on the tasks.

Critical time can be determined by looking at the longest sequence of tasks in the digraph, called the critical path

Adding an algorithm

Up until now, we have been creating schedules by guess-and-check, which works well enough for small schedules, but would not work well with dozens or hundreds of tasks. To create a more procedural approach, we might begin by somehow creating a **priority list**. Once we have a priority list, we can begin scheduling using that list and the **list processing algorithm**.

Priority List
A priority list is a list of tasks given in the order in which we desire them to be completed.

The List Processing Algorithm turns a priority list into a schedule

List Processing Algorithm
1. On the digraph or priority list, circle all tasks that are ready, meaning that all pre-requisite tasks have been completed.
2. Assign to each available processor, in order, the first ready task. Mark the task as in progress, perhaps by putting a single line through the task.
3. Move forward in time until a task is completed. Mark the task as completed, perhaps by crossing out the task. If any new tasks become ready, mark them as such.
4. Repeat until all tasks have been scheduled.

Example 2

Using our digraph from above, schedule it using the priority list below:
$T_1, T_3, T_4, T_5, T_6, T_7, T_8, T_2, T_9$

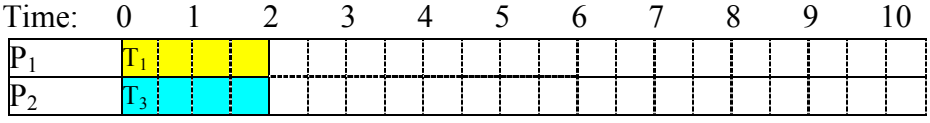
Time 0: Mark ready tasks

Priority list: (T1) (T3) (T4) (T5) T6, T7, T8, T2, T9

We assign the first task, T_1 to the first processor, P_1, and the second ready task, T_3, to the second processor. Making those assignments, we mark those tasks as in progress:

Priority list: ~~(T1)~~ ~~(T3)~~ (T4) (T5) T6, T7, T8, T2, T9

Schedule up to here:

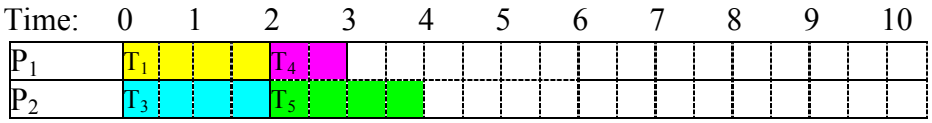


We jump to the time when the next task completes, which is at time 2.

Time 2: Both processors complete their tasks. We mark those tasks as complete. With Task 1 complete, Task 2 becomes ready:

Priority list: ~~T_1~~, ~~T_3~~, T_4, T_5, T_6, T_7, T_8, T_2, T_9
We assign the next ready task on the list, T_4 to P_1, and T_5 to P_2.

Priority list: ~~T_1~~, ~~T_3~~, ~~T_4~~, ~~T_5~~, T_6, T_7, T_8, T_2, T_9

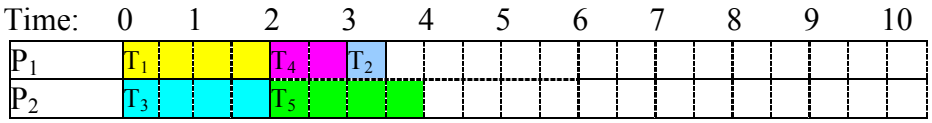


Time 3: Processor 1 has completed T_4. Completing T_4 does not make any other tasks ready (note that all the rest require that T_2 be completed first).

Priority list: ~~T_1~~, ~~T_3~~, ~~T_4~~, ~~T_5~~, T_6, T_7, T_8, T_2, T_9

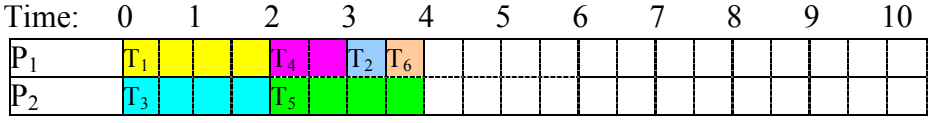
Since the next three tasks are not yet ready, we assign the next ready task, T_2 to P_1

Priority list: ~~T_1~~, ~~T_3~~, ~~T_4~~, ~~T_5~~, T_6, T_7, T_8, ~~T_2~~, T_9



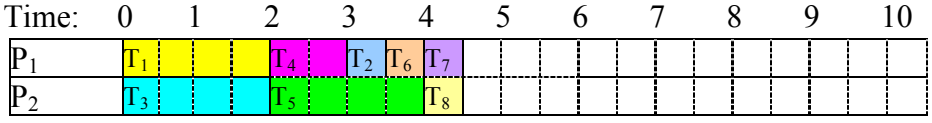
Time 3.5: Processor 1 has completed T_2. Completing T_2 causes T_6 and T_7 to become ready. We assign T_6 to P_1

Priority list: ~~T_1~~, ~~T_3~~, ~~T_4~~, ~~T_5~~, ~~T_6~~, ~~T_7~~, T_8, ~~T_2~~, T_9



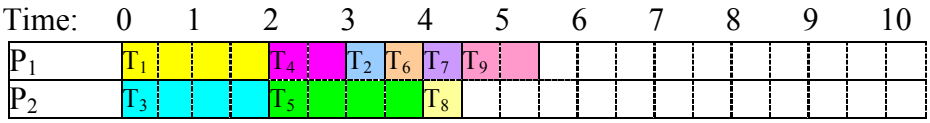
Time 4: Both processors complete their tasks. The completion of T_5 allows T_8 to become ready. We assign T_7 to $P_1$and T_8 to P_2.

Priority list: ~~T_1~~, ~~T_3~~, ~~T_4~~, ~~T_5~~, ~~T_6~~, ~~T_7~~, ~~T_8~~, ~~T_2~~, T_9



Time 4.5: Both processors complete their tasks. T_9 becomes ready, and is assigned to P_1. There is no ready task for P_2 to work on, so P_2 idles.

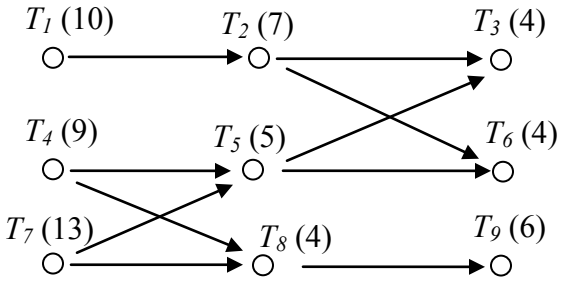
Priority list: ~~T_1~~, ~~T_3~~, ~~T_4~~, ~~T_5~~, ~~T_6~~, ~~T_7~~, ~~T_8~~, ~~T_2~~, ~~T_9~~



With the last task completed, we have a completed schedule, with finishing time 5.5 days.

Try it Now 1

Using the digraph below, create a schedule using the priority list:
$T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9$



It is important to note that the list processing algorithm itself does not influence the resulting schedule – the schedule is completely determined by the priority list followed. The list processing, while do-able by hand, could just as easily be executed by a computer. The interesting part of scheduling, then, is how to choose a priority list that will create the best possible schedule.

Choosing a priority list

We will explore two algorithms for selecting a priority list.

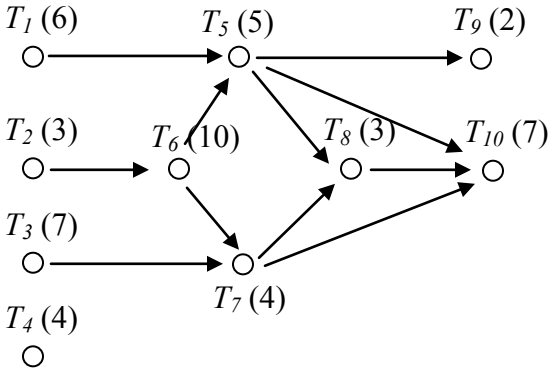
Decreasing time algorithm

The decreasing time algorithm takes the approach of trying to get the very long tasks out of the way as soon as possible by putting them first on the priority list.

Decreasing Time Algorithm
Create the priority list by listing the tasks in order from longest completion time to shortest completion time.

Example 3

Consider the scheduling problem represented by the digraph below. Create a priority list using the decreasing time list algorithm, then use it to schedule for two processors using the list processing algorithm.



To use the decreasing time list algorithm, we create our priority list by listing the tasks in order from longest task time to shortest task time. If there is a tie, we will list the task with smaller task number first (not for any good reason, but just for consistency).

For this digraph, the decreasing time algorithm would create a priority list of:

T_6 (10), T_3 (7), T_{10} (7), T_1 (6), T_5 (5), T_4 (4), T_7 (4), T_2 (3), T_8 (3), T_9 (2)

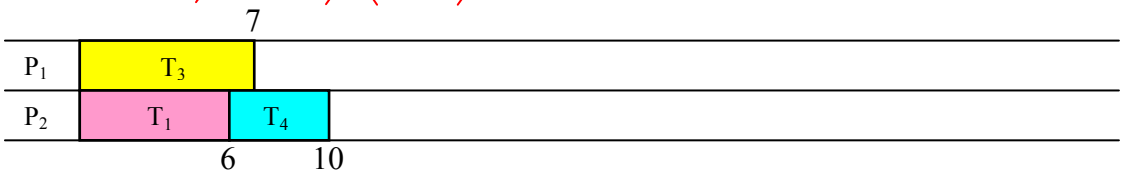
Once we have the priority list, we can create the schedule using the list processing algorithm. With two processors, we'd get:

Time 0: We identify ready tasks, and assign T_3 to P_1 and T_1 to P_2

Priority list: T_6, ~~T_3~~, T_{10}, ~~T_1~~, T_5, ~~T_4~~, T_7, ~~T_2~~, T_8, T_9

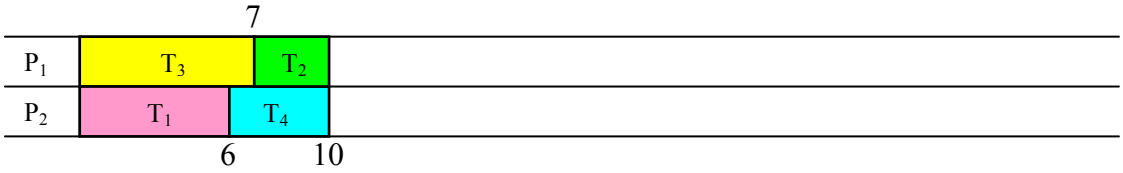
Time 6: P_2 completes T_1. No new tasks become ready, so T_4 is assigned to P_2.

Priority list: T_6, ~~T_3~~, T_{10}, ~~T_1~~, T_5, ~~T_4~~, T_7, ~~T_2~~, T_8, T_9



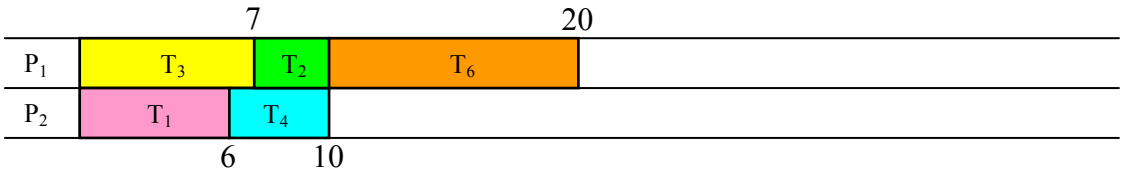
Time 7: P_1 completes T_3. No new tasks become ready, so T_2 is assigned to P_1.

Priority list: T_6, ~~T_3~~, T_{10}, ~~T_1~~, T_5, ~~T_4~~, T_7, ~~T_2~~, T_8, T_9



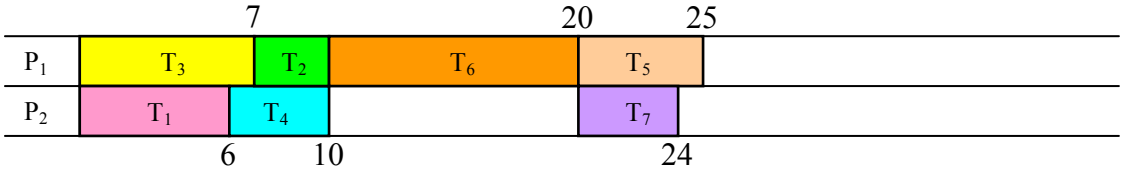
Time 10: Both processors complete their tasks. T_6 becomes ready, and is assigned to P_1. No other tasks are ready, so P_2 idles.

Priority list: ~~T_6~~, ~~T_3~~, T_{10}, ~~T_1~~, T_5, ~~T_4~~, T_7, ~~T_2~~, T_8, T_9



Time 20: With T_6 complete, T_5 and T_7 become ready, and are assigned to P_1 and P_2 respectively.

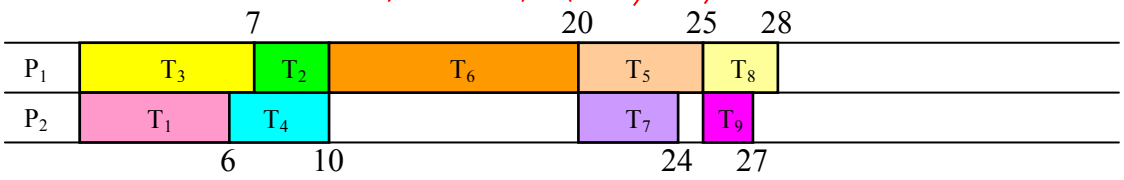
Priority list: ~~T_6~~, ~~T_3~~, T_{10}, ~~T_1~~, ~~T_5~~, ~~T_4~~, ~~T_7~~, ~~T_2~~, T_8, T_9



Time 24: P_2 completes T_7. No new items become ready, so P_2 idles.

Time 25: P_1 completes T_5. T_8 and T_9 become ready, and are assigned.

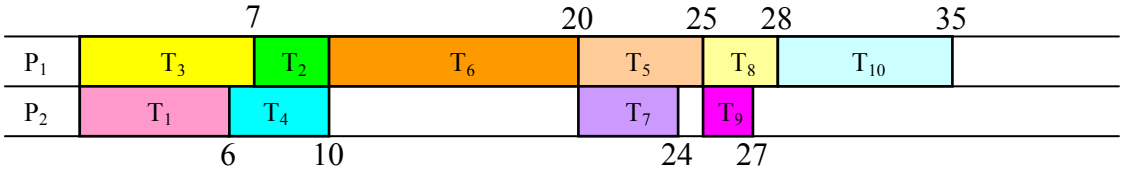
Priority list: ~~T_6~~, ~~T_3~~, T_{10}, ~~T_1~~, ~~T_5~~, ~~T_4~~, ~~T_7~~, ~~T_2~~, ~~T_8~~, ~~T_9~~



Time 27: T_9 is completed. No items ready, so P_2 idles.

Time 28: T_8 is completed. T_{10} becomes ready, and is assigned to P_1.

Priority list: ~~T_6~~, ~~T_3~~, ~~T_{10}~~, ~~T_1~~, ~~T_5~~, ~~T_4~~, ~~T_7~~, ~~T_2~~, ~~T_8~~, ~~T_9~~



This is our completed schedule, with a finishing time of 35.

Using the decreasing time algorithm, the priority list led to a schedule with a finishing time of 35. Is this good? It certainly looks like there was a lot of idle time in this schedule. To get some idea how good or bad this schedule is, we could compute the critical time, the minimum time to complete the job. To find this, we look for the sequence of tasks with the highest total completion time. For this digraph that sequence would appear to be: T_2, T_6, T_5, T_8, T_{10}, with total sequence time of 28. From this we can conclude that our schedule isn't horrible, but there is a possibility that a better schedule exists.

Try it Now 2
Determine the priority list for the digraph from Try it Now 1 using the decreasing time algorithm.

Critical path algorithm

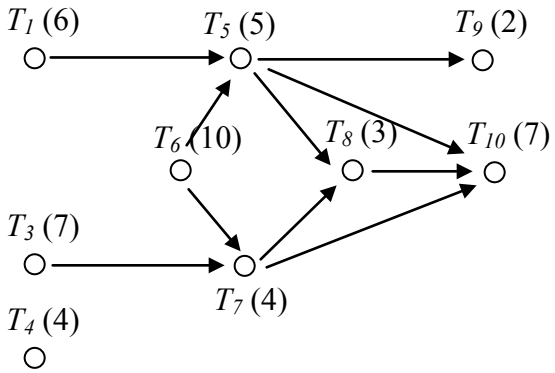
A sequence of tasks in the digraph is called a **path**. In the previous example, we saw that the critical path dictates the minimum completion time for a schedule. Perhaps, then, it would make sense to consider the critical path when creating our schedule. For example, in the last schedule, the processors began working on tasks 1 and 3 because they were longer tasks, but starting on task 2 earlier would have allowed work to begin on the long task 6 earlier.

The critical path algorithm allows you to create a priority list based on idea of critical paths.

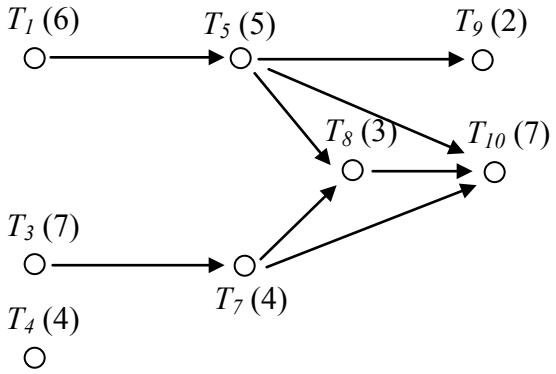
Critical Path Algorithm (version 1)
1. Find the critical path.
2. The first task in the critical path gets added to the priority list.
3. Remove that task from the digraph
4. Repeat, finding the new critical path with the revised digraph.

Example 4

The original digraph from Example 3 has critical path T_2, T_6, T_5, T_8, T_{10}, so T_2 gets added first to the priority list. Removing T_2 from the digraph, it now looks like:



The critical path (longest path) in the remaining digraph is now T_6, T_5, T_8, T_{10}, so T_6 is added to the priority list and removed.

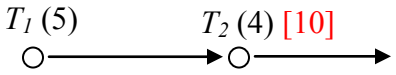


Now there are two paths with the same longest length: T_1, T_5, T_8, T_{10} and T_3, T_7, T_8, T_{10}. We can add T_1 to the priority list (or T_3 – we usually add the one with smaller item number) and remove it, and continue the process.

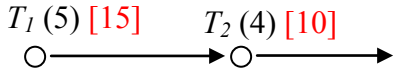
I'm sure you can imagine that searching for the critical path every time you remove a task from the digraph would get really tiring, especially for a large digraph. In practice, the critical path algorithm is implementing by first working from the end backwards. This is called the **backflow algorithm**.

Backflow Algorithm

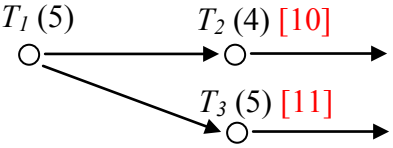
1. Introduce an "end" vertex, and assign it a time of 0, shown in [brackets]
2. Move backwards to every vertex that has an arrow to the end and assign it a critical time
3. From each of those vertices, move backwards and assign those vertices critical times. Notice that the critical time for the earlier vertex will be that task's time plus the critical time for the later vertex.
   Example: Consider this segment of digraph.



   In this case, if T2 has already been determined to have a critical time of 10, then T1 will have a critical time of 5+10 = 15



   If you have already assigned a critical time to a vertex, replace it only if the new time is larger.
   Example: In the digraph below, T1 should be labeled with a critical time of 16, since it is the longer of 5+10 and 5+11.



4. Repeat until all vertices are labeled with their critical times

One you have completed the backflow algorithm, you can easily create the critical path priority list by using the critical times you just found.

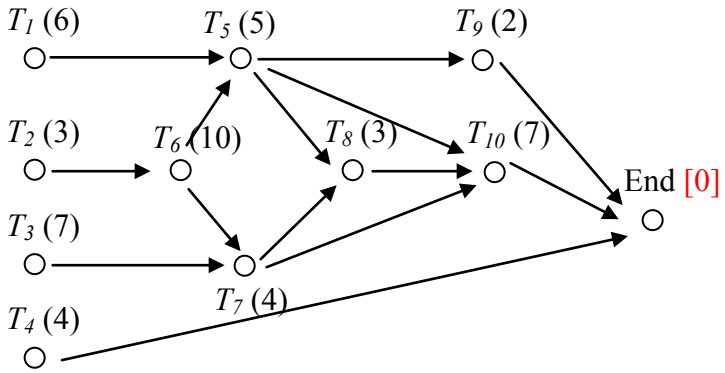
Critical Path Algorithm (version 2)
1. Apply the backflow algorithm to the digraph
2. Create the priority list by listing the tasks in order from longest critical time to shortest critical time

This version of the Critical Path Algorithm will usually be the easier to implement.

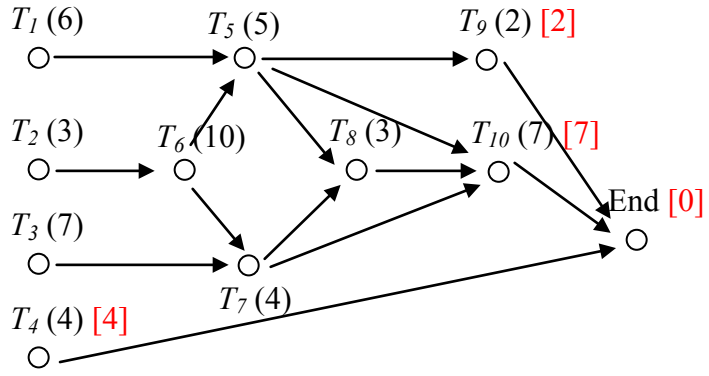
Example 5

Applying this to our digraph from the earlier example, we start applying the backflow algorithm.

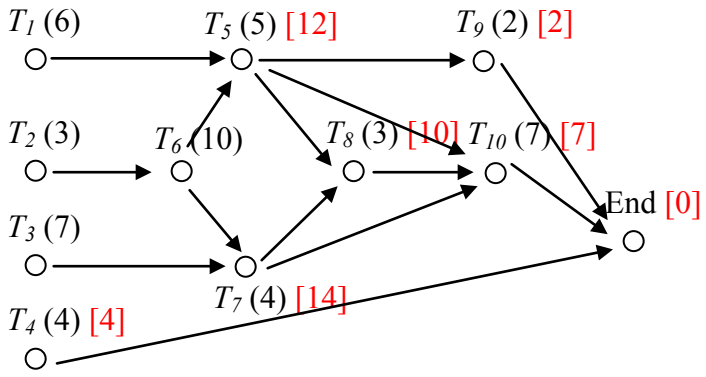
We add an end vertex and give it a critical time of 0.



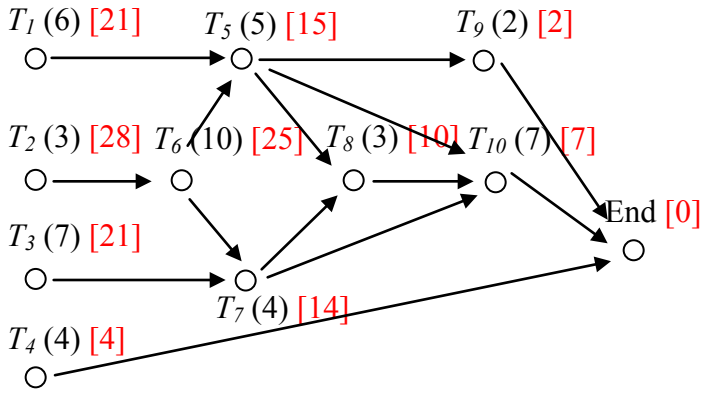
We then move back to T_4, T_9, and T_{10}, labeling them with their critical times



From each vertex marked with a critical time, we go back. T_7, for example, will get labeled with a critical time 11 – the task time of 4 plus the critical time of 7 for T_{10}. For T_5, there are two paths to the end. We use the longer, labeling T_5 with critical time 5+7 = 12.



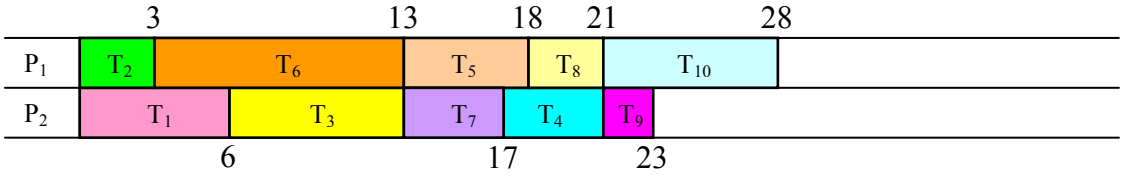
Continue the process until all vertices are labeled. Notice that the critical time for T_5 ended got replaced later with the even longer path through T_8.



We can now quickly create the critical path priority list by listing the tasks in decreasing order of critical time:

Priority list: T_2, T_6, T_1, T_3, T_5, T_7, T_8, T_{10}, T_4, T_9

Applying this priority list using the list processing algorithm, we get the schedule:



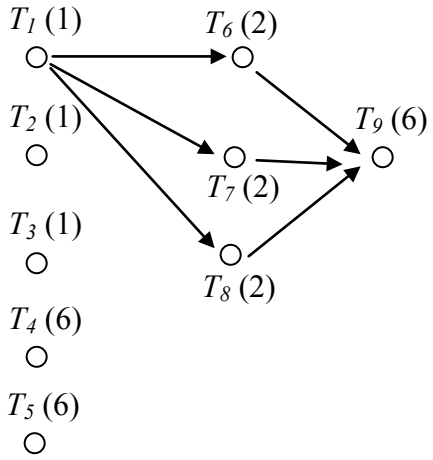
In this particular case, we were able to achieve the minimum possible completion time with this schedule, suggesting that this schedule is optimal. This is certainly not always the case.

Try it Now 3

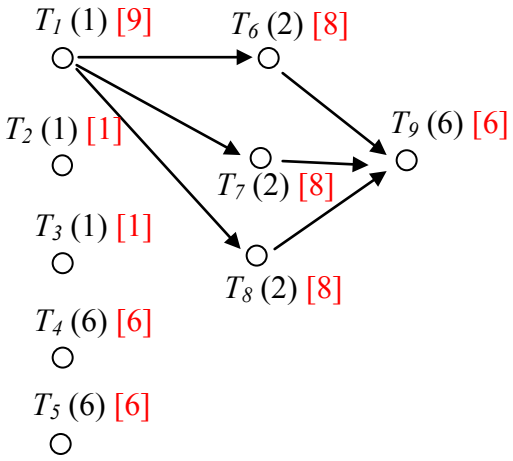
Determine the priority list for the digraph from Try it Now 1 using the critical path algorithm.

Example 6

This example is designed to show that the critical path algorithm doesn't always work wonderfully. Schedule the tasks in the digraph below on three processors using the critical path algorithm.

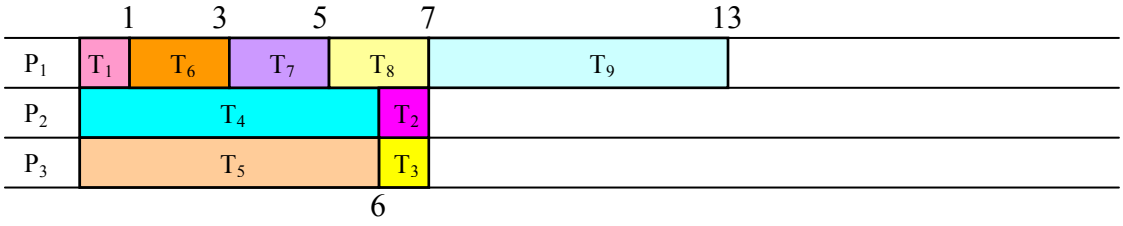


To create a critical path priority list, we could first apply the backflow algorithm:



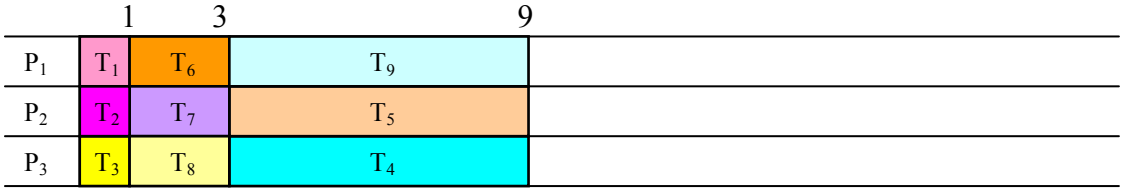
This yields the critical-path priority list: T_1, T_6, T_7, T_8, T_4, T_5, T_9, T_2, T_3.

Applying the list processing algorithm to this priority list leads to the schedule:



This schedule has finishing time of 13.

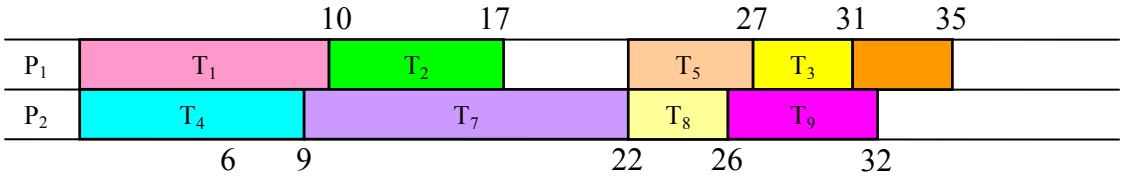
By observation, we can see that a much better schedule exists for the example above:



In most cases the critical path algorithm will lead to a very good schedule. There are cases, like this, where it will not. Unfortunately, there is no known algorithm to always produce the optimal schedule.

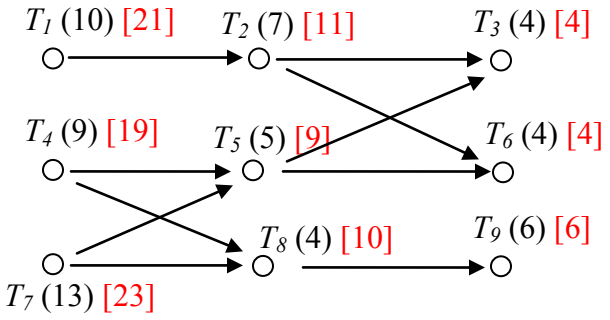
Try it Now Answers

1. Time 0: Tasks 1, 4, and 7 are ready. Assign Tasks 1 and 4
Time 9: Task 4 completes. Only task 7 is ready; assign task 7
Time 10: Task 1 completes. Task 2 now ready. Assign task 2
Time 17: Task 2 completes. Nothing is ready. Processor 1 idles
Time 22: Task 7 completes. Tasks 5 and 8 are ready. Assign tasks 5 and 8
Time 26: Task 8 completes. Task 9 is ready. Assign task 9
Time 27: Task 5 completes. Tasks 3 and 6 are ready. Assign task 3
Time 31: Task 3 completes. Assign task 6
Time 35: Everything is done. Finishing time is 35 for this schedule.



2. T_7, T_1, T_4, T_2, T_9, T_5, T_3, T_6, T_8

3. Applying the backflow algorithm, we get this:



The critical path priority list is: T_7, T_1, T_4, T_2, T_8, T_5, T_9, T_3, T_6

Exercises

Skills

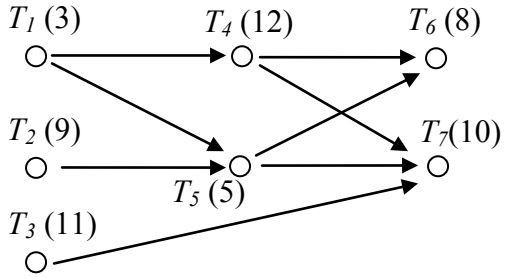
1. Create a digraph for the following set of tasks:

Task	Time required	Tasks that must be completed first
A	3	
B	4	
C	7	
D	6	A, B
E	5	B
F	5	D, E
G	4	E

2. Create a digraph for the following set of tasks:

Task	Time required	Tasks that must be completed first
A	3	
B	4	
C	7	
D	6	A
E	5	A
F	5	B
G	4	D, E

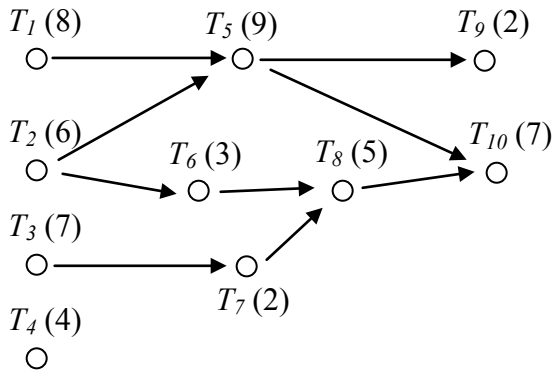
Use this digraph for the next 2 problems.



3. Using the priority list T_4, T_1, T_7, T_3, T_6, T_2, T_5, schedule the project with two processors.

4. Using the priority list T_5, T_2, T_3, T_7, T_1, T_4, T_6, schedule the project with two processors.

Use this digraph for the next 4 problems.



5. Using the priority list T_4, T_3, T_9, T_{10}, T_8, T_5, T_6, T_1, T_7, T_2 schedule the project with two processors.

6. Using the priority list T_2, T_4, T_6, T_8, T_{10}, T_1, T_3, T_5, T_7, T_9 schedule the project with two processors.

7. Using the priority list T_4, T_3, T_9, T_{10}, T_8, T_5, T_6, T_1, T_7, T_2 schedule the project with three processors.

8. Using the priority list T_2, T_4, T_6, T_8, T_{10}, T_1, T_3, T_5, T_7, T_9 schedule the project with three processors.

9. Use the decreasing time algorithm to create a priority list for the digraph from #3, and schedule with two processors.

10. Use the decreasing time algorithm to create a priority list for the digraph from #3, and schedule with three processors.

11. Use the decreasing time algorithm to create a priority list for the digraph from #5, and schedule with two processors.

12. Use the decreasing time algorithm to create a priority list for the digraph from #5, and schedule with three processors.

13. Use the decreasing time algorithm to create a priority list for the problem from #1, and schedule with two processors.

14. Use the decreasing time algorithm to create a priority list for the problem from #2, and schedule with two processors.

15. With the digraph from #3:
    a. Apply the backflow algorithm to find the critical time for each task
    b. Find the critical path for the project and the minimum completion time
    c. Use the critical path algorithm to create a priority list and schedule on two processors.

16. With the digraph from #3, use the critical path algorithm to schedule on three processors.

17. With the digraph from #5:
    a. Apply the backflow algorithm to find the critical time for each task
    b. Find the critical path for the project and the minimum completion time
    c. Use the critical path algorithm to create a priority list and schedule on two processors.

18. With the digraph from #5, use the critical path algorithm to schedule on three processors.

19. Use the critical path algorithm to schedule the problem from #1 on two processors.

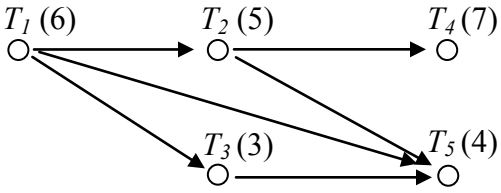
20. Use the critical path algorithm to schedule the problem from #2 on two processors.

Concepts

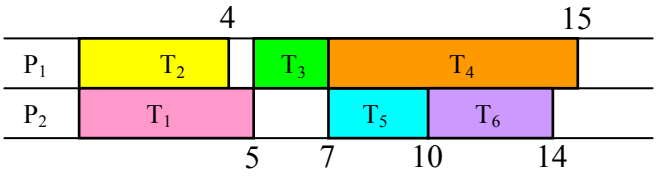
21. If an additional order requirement is added to a digraph, can the optimal finishing time ever become longer? Can the optimal finishing time ever become shorter?

22. Will an optimal schedule always have no idle time?

23. Consider the digraph below.
    a. How many priority lists could be created for these tasks?
    b. How many unique schedules are created by those priority lists?



24. Create a digraph and priority list that would lead to the schedule below.



25. Is it possible to create a digraph with three tasks for which every possible priority list creates a different schedule? If so, create it.

26. Is it possible to create a digraph with four tasks for which every possible priority list creates a different schedule? If so, create it.

Exploration

27. Independent tasks are ones that have no order requirements; they can be completed in any order.
    a. Consider three tasks, with completion times 2, 2, and 4 hours respectively. Construct two different schedules on two processors with different completion times to show that the priority list still matters with independent tasks.
    b. Choose a set of independent tasks with different completion times, and implement the decreasing time list algorithm and the critical path algorithm. What do you observe?
    c. Will using the decreasing time list or critical path algorithms with independent tasks always produce an optimal schedule? Why or why not?
    d. Will using the decreasing time list or critical path algorithms with independent tasks always produce the same schedule? Why or why not?

28. In a group, choose ten tasks necessary to throw a birthday party for a friend or child (for example, cleaning the house or buying a cake). Determine order requirements for the tasks, create a digraph, and schedule the tasks for two people.

29-37: At the end of the chapter it was noted that no algorithm exists to determine if an arbitrary schedule is optimal, but there are special cases where we can determine that a schedule is indeed optimal. In each of the following scenarios, determine
    a) If the scenario is even possible
    b) Whether or not the schedule *could* be optimal
    c) Whether or not we can be *sure* that the schedule is optimal

29. A job has a critical time of 30 hours, and the finishing time for the schedule on 2 processors is 30 hours.
30. The sum of all task times for a job is 40 hours, and the finishing time for the schedule on 2 processors was 15 hours
31. The sum of all task times for a job is 100 hours, the critical time of the job was 40 hours, and the finishing time for the schedule on 2 processors was 50 hours.
32. The sum of all task times for a job is 50 hours, and the finishing time for the schedule on 2 processors was 40 hours.
33. A job has a critical time of 30 hours, and the finishing time for the schedule on 3 processors is 20 hours.
34. The sum of all task times for a job is 60 hours, and the finishing time for the schedule on 3 processors was 20 hours.
35. The critical time for a job is 25 hours, and the finishing time for the schedule on 2 processors was 30 hours.
36. The sum of all task times for a job is 20 hours, and the finishing time for the schedule on 2 processors was 25 hours.
37. Based on your observations in the previous scenarios, write guidelines for when you can determine that a schedule is optimal.