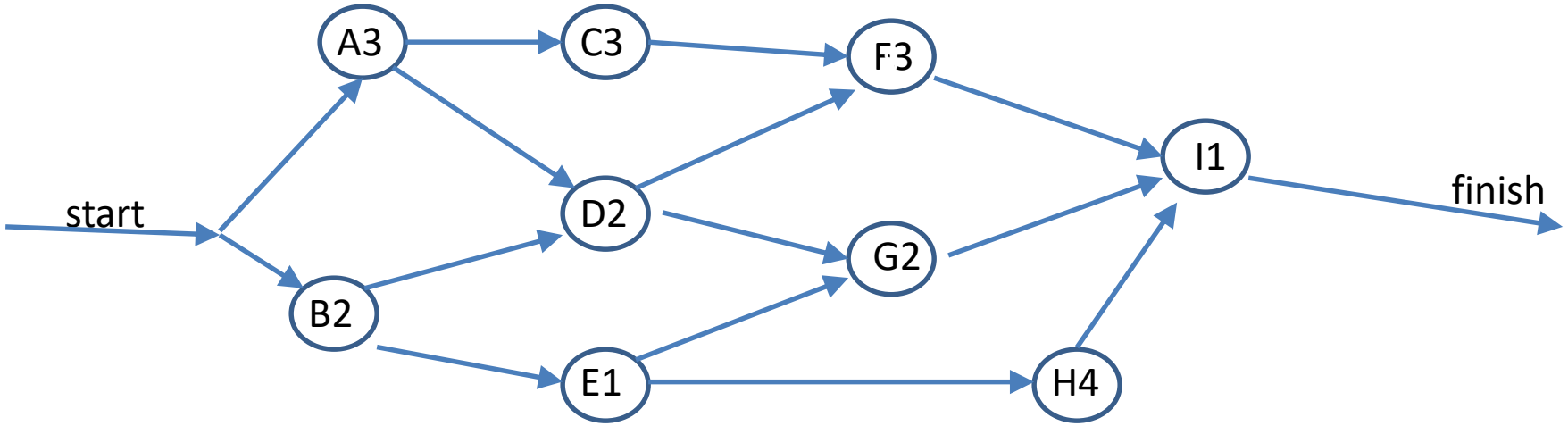
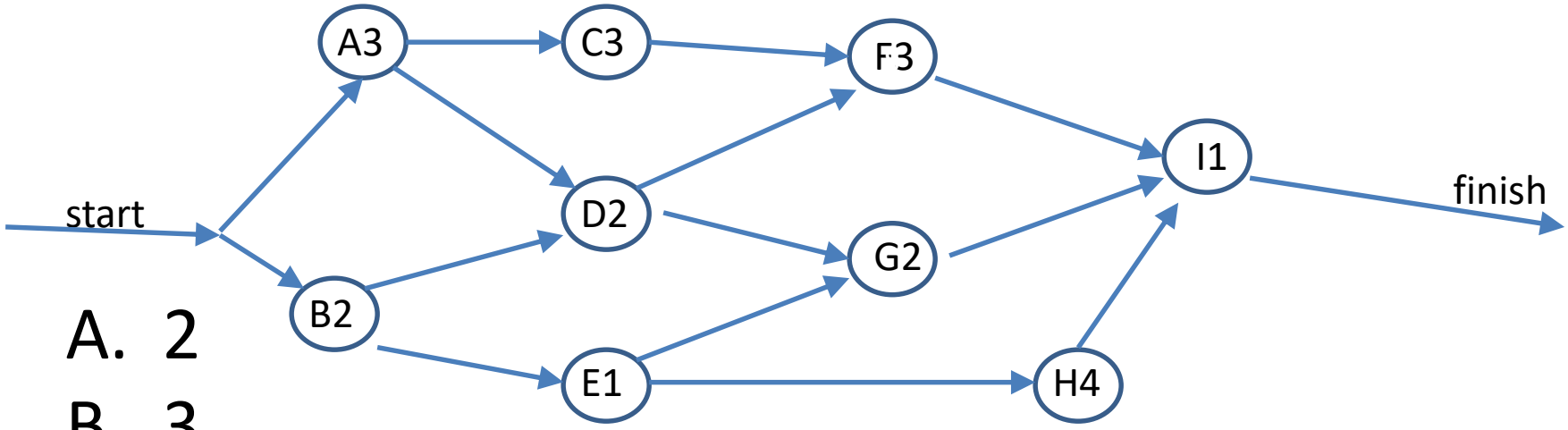
Activity Graphs

An *activity-node graph* has nodes that represent activities and the time they take to complete. An edge such as

indicates that X must be completed before Y begins. X takes time t1 and Y takes time t2. Here is a typical activity graph:



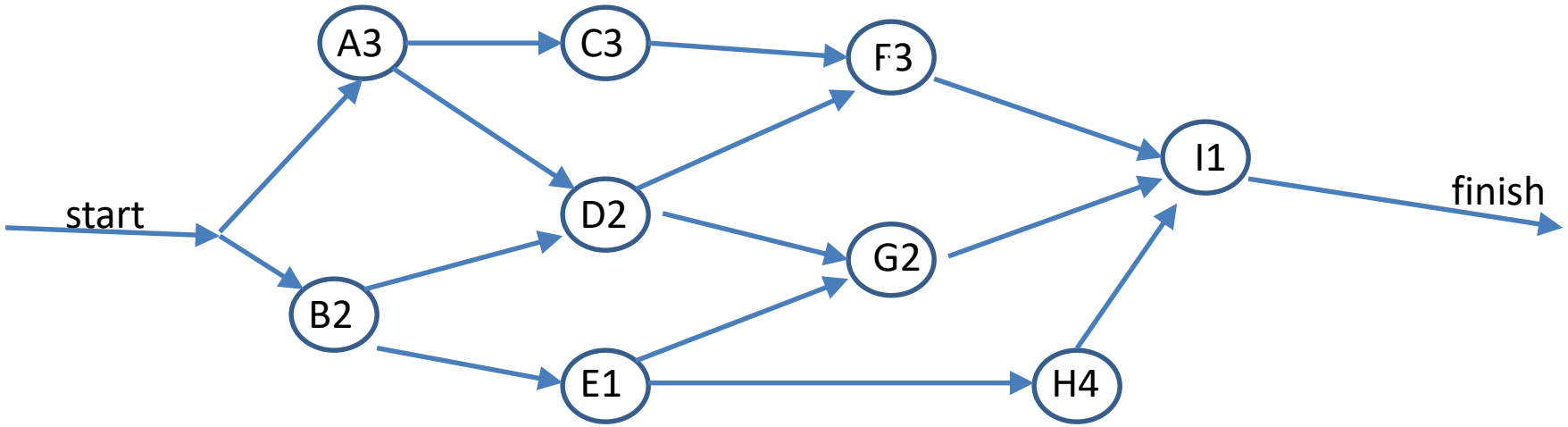
Question: Here is the kind of information we'd like to get from this graph: If we start both A and B at time 0, what is the earliest time we could complete D?



A. 2
B. 3
C. 4
D. 5
E. 7

Answer D: time 5

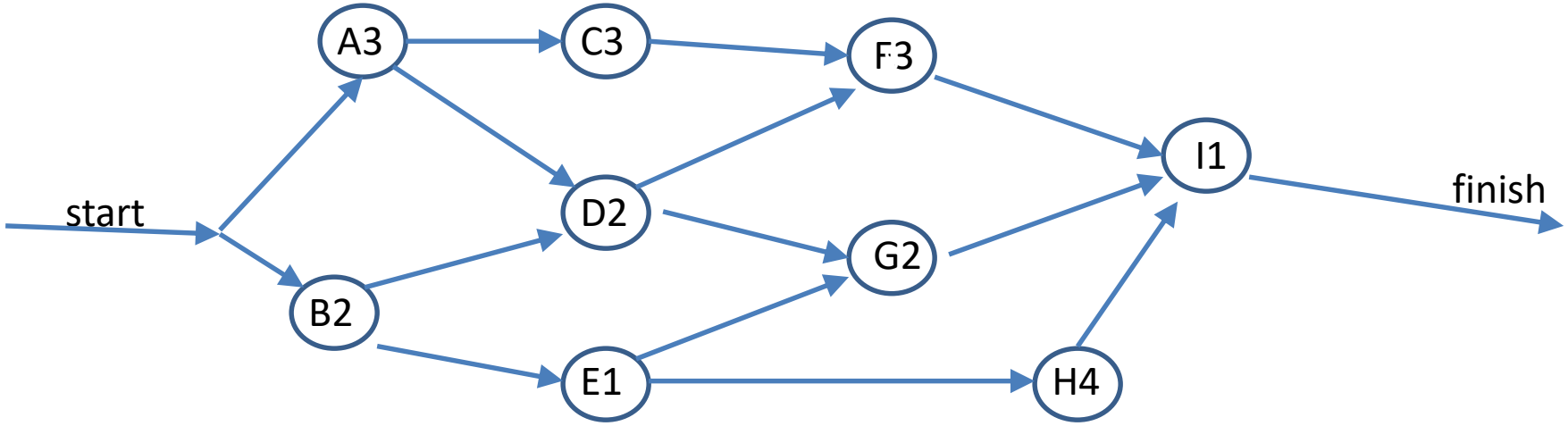
Question: If the nodes on this activity graph represent the parts of a project, how long will it take to complete the project?



A. 6
B. 8
C. 10
D. 21 (the sum of all the times)

Answer C: 10

One more: Suppose part D takes time 3 instead of time 2. Will that delay the completion time of part F?



A. No
B. Yes

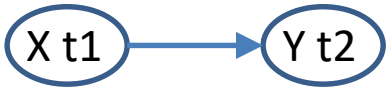
We will find algorithms that answer questions like this.

Answer A: No

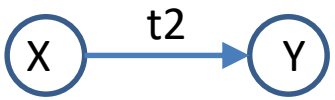
It would be nice to have algorithms for answering questions like this. We don't have any algorithms for exploring graphs where the costs are in the nodes themselves, so we turn this into an *event-node graph* in which the nodes represent the completion of an event and the edges represent the time the event takes.

Here is an algorithm for the conversion:

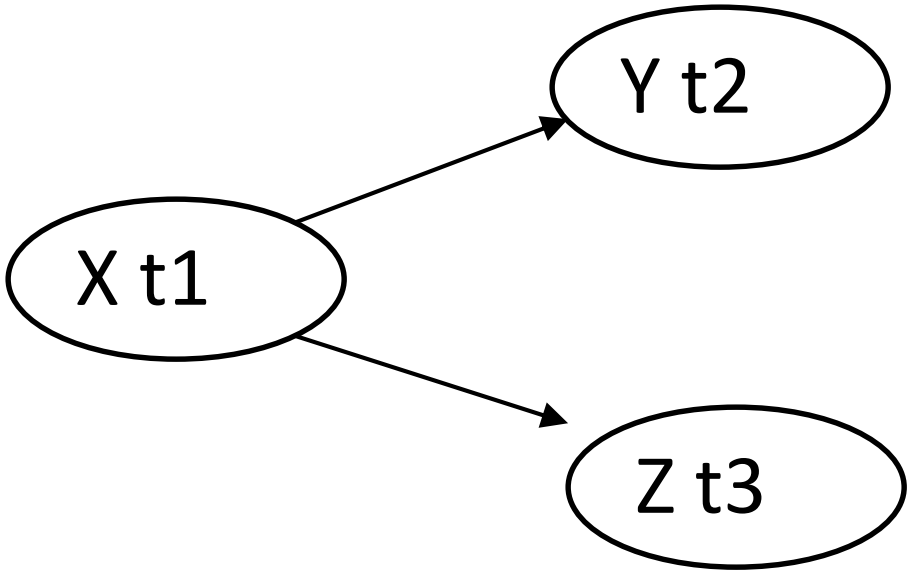
A. If Y has only one incoming edge in the event graph:



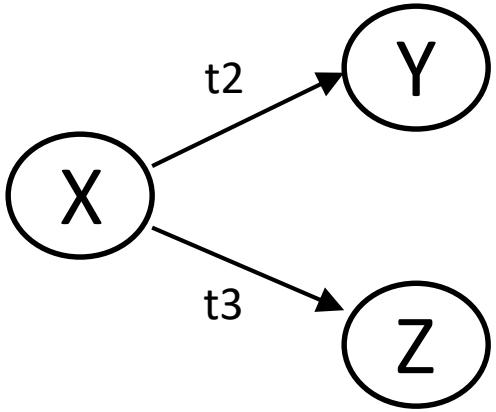
replace this edge by one with cost t2 in the event graph:



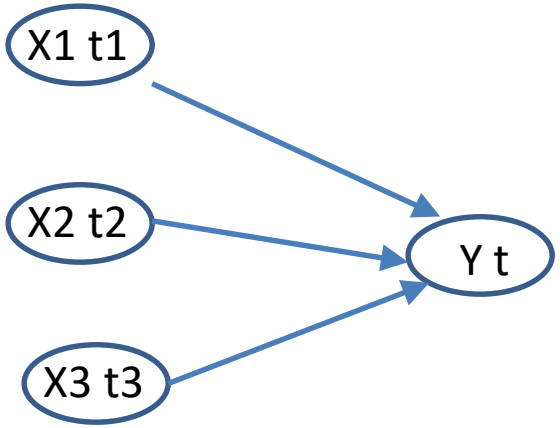
Similarly, if there are several edges going out from X



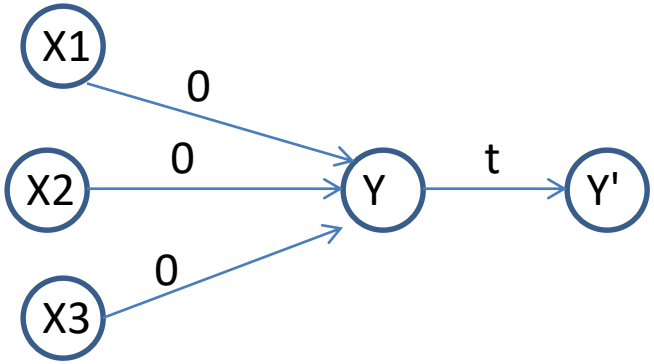
put the weights of the destinations on the edges:



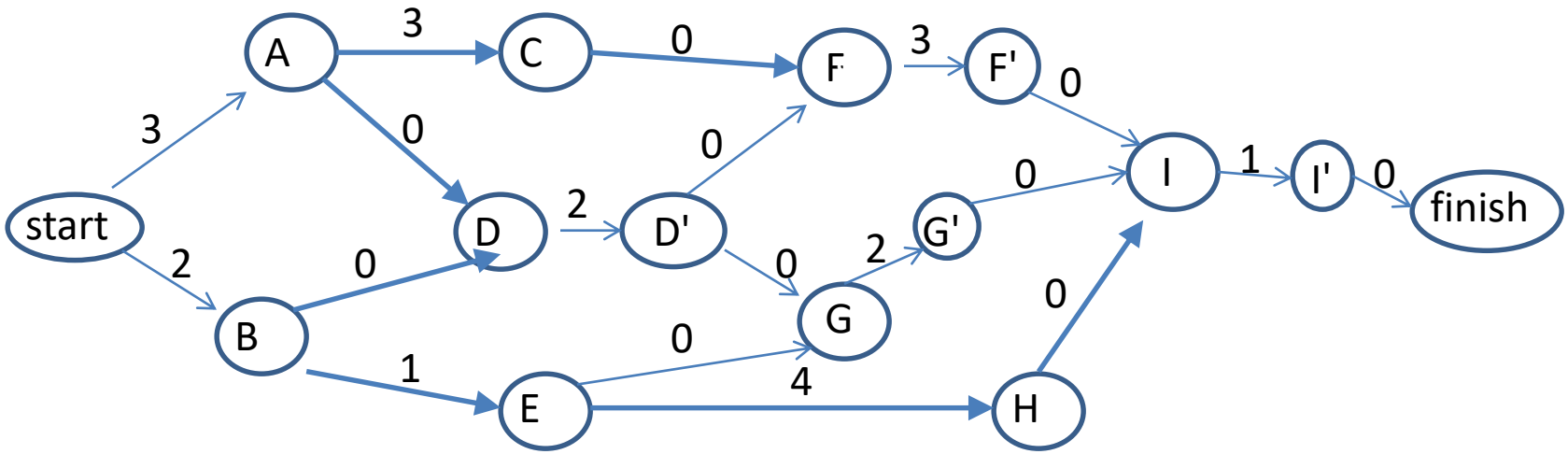
B. If Y has multiple incoming edges in the activity graph,



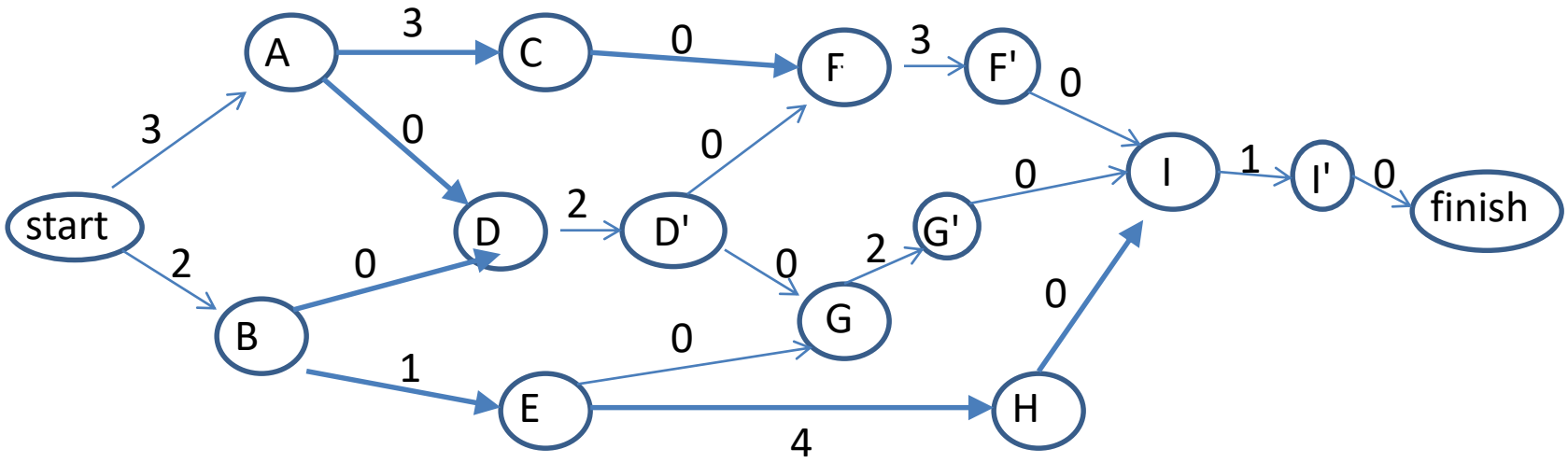
make the corresponding edges in the event graph have cost 0, split Y into 2 nodes Y and Y', and make the edge from Y to Y' have weight t



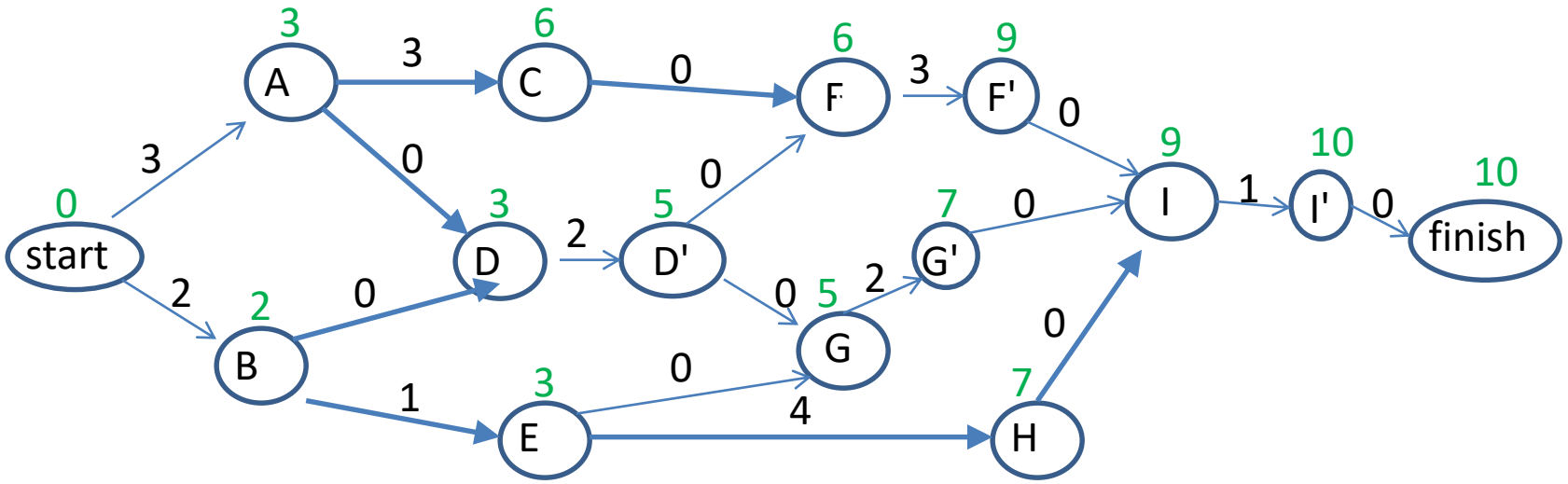
With these rules our activity graph becomes an event graph – the nodes represent an event, the completion of an activity:



Question: How can we find the *Earliest Completion Time* (EC) of a node – the soonest it could be completed if our times are correct? For example, what is the EC of node D? Of node G?

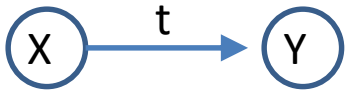


The *longest* path from start to a node gives the earliest possible completion time of the node. We can easily modify the Bellman-Ford shortest-path algorithm to give the longest path in the case of an acyclic graph -- just reverse the inequalities.



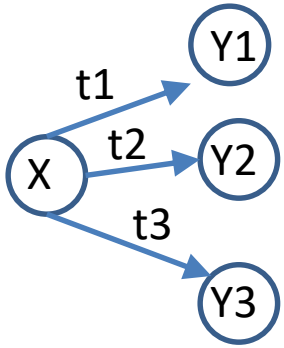
The Earliest Completion of each node is shown on the graph in green.

We can also compute the Latest Completion time for each node, which is the last time the node can be completed without delaying the project's overall completion time. We start at the finish node and work backwards. The LC and EC times for the finish node a are the same. For other nodes, if we have



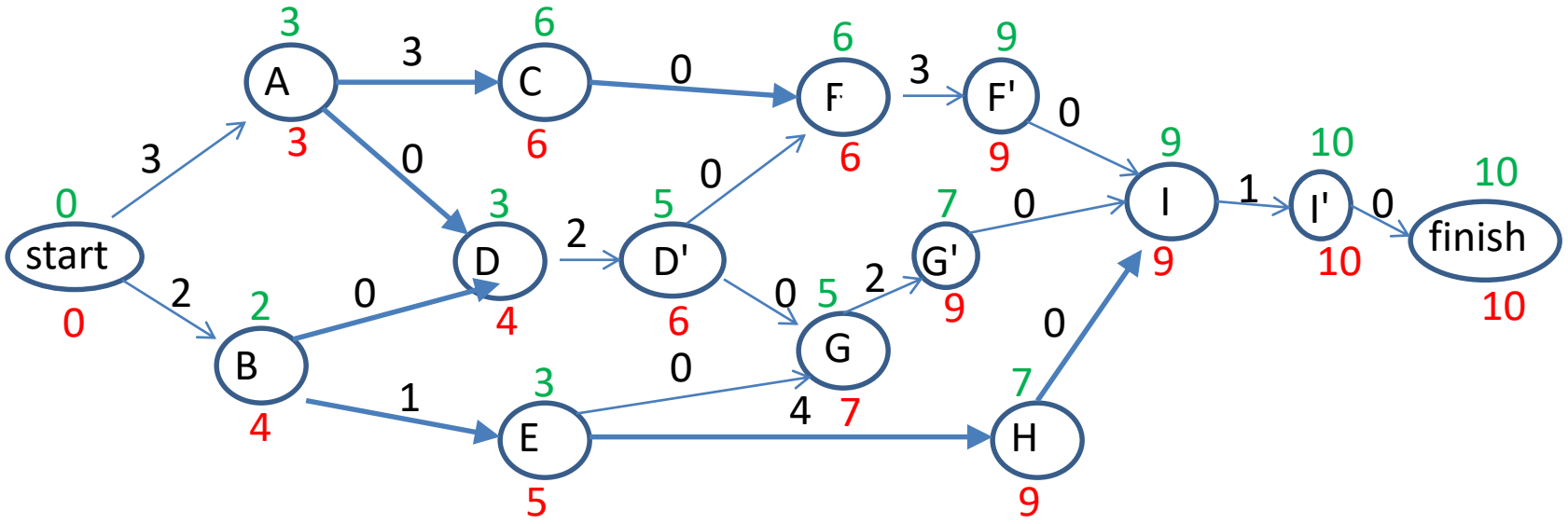
then X.LC = Y.LC - t

If we have



then X.LC = min{Y1.LC-t1, Y2.LC-t2, Y3.LC-t3}

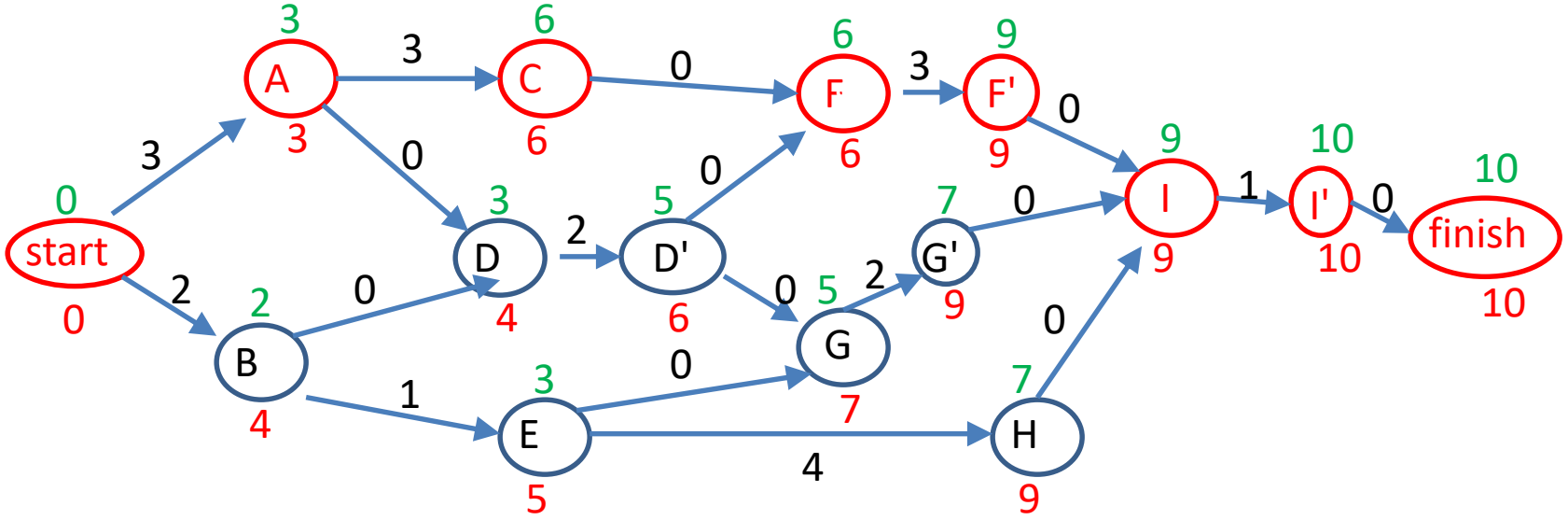
Here is our event graph with the LC times written in red:



For any node the difference between the earliest time it can be completed and the latest time it must be completed by to avoid delaying the project is called that activity's *slack time:*

Slack = LC - EC

On the longest path through the graph, EC = LC at every node, so the slack time is 0. This is called the *critical path* for the graph. Any delay on the critical path delays the whole project. Much project planning goes into ensuring that activities on the critical path stay on schedule.



The critical path is shown in red.