The Critical Path Method

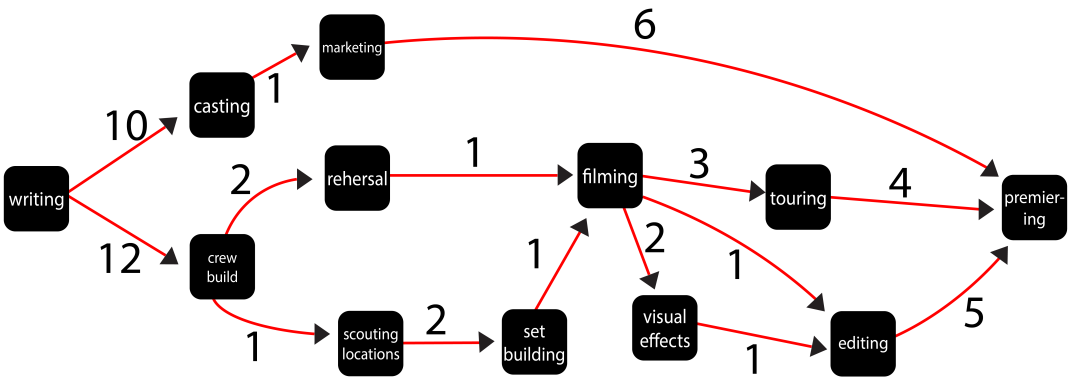
William Clark and Henry Howell

April 30, 2021

This paper demonstrates the critical path method with an example, using the framework for the method provided by Upasana Chatterjee in his paper, "Applications of Three Common Graph Theoretical Algorithms." This paper will walk through the critical path method by way of a hypothetical example. This example will walk through the steps necessary to finding the critical path and will find the earliest start times and latest possible start times for each task in the project plan. Consider this scenario, William and Henry are planning to make a movie, but aren't entirely sure in what order of events they will run their production. Our goal is to optimize the process by devoting more effort towards those tasks that will take the longest while also seeing moments in which we can streamline and run tasks in sync. We will use the critical path method to take us all the way from the nascency of writing the script to the nerve-wracking moment of premiering it in front of, hopefully, a large audience. In more technical terms, the source of this process is the onset of our script while the final event is the day we show it to audiences. The critical path method will make sure we stay on track to premier on the correct date and shows us where we should focus our efforts.

Creating Directed Acyclic Task Graph

The first step in solving for a critical path is the creation of a directed acyclic task graph. To do so, we listed out all the tasks needed to get from the writing stage to the premiering stage. We weighted each of these tasks, as can be seen in the figure below, based on how long they would take before the next task can start. Some of the tasks are not dependent on the previous task being completed. For example, casting can start after writing has been going on for 10 months, but crew building cannot start until writing has been completed after 12 months. The most important part of this graph, is obviously that it is directed but also that it is acyclic. There can be no cycles, because the graph must start at the writing phase and end with the premiering of the movie. In order to ensure the graph is a directed acyclic graph (DAG), the events must have a topological ordering, something that we will go into in the next section.



Picking a Topological Ordering

We have our directed acyclic graph G, and now our goal is to topologically order the nodes in G. More concretely, we seek a nonnegative distinct integer index for each task such that whenever task j must begin after task i begins, we have $i < j$. Here, we will introduce the notation of $X_i s$ for each task in our task graph, where an X_i will represent a certain task in the making of the movie. To determine a topological ordering, we begin by looking at the tasks in the task graph that have no incoming edges. The only one is writing. Let us pause here to provide some reasoning as to why there must always be a node without incoming edges in a DAG task graph.

Proof: We will do a proof by contradiction. Let $G = (V, E)$ with $|V| = n$ be an arbitrary directed acyclic graph such that every node $v \in V$ has at least one incoming edge. Furthermore, let (v_i, v_j) for $i \neq j$ be an ordered pair representing a directed edge in E. Hence, for each $v_j \in V$, there is a $v_i \in V$ with $i \neq j$ such that $(v_i, v_j) \in E$. We claim that there is a cycle in G, a contradiction. To see this, begin at an arbitrary node in G, say a. We will now construct a path by moving against the direction of the directed edges. We know that there must be another node b such that $(b, a) \in E$. Likewise, b has an incoming edge, so it must be either one of the remaining $n - 2$ nodes, or a. Since each node has an incoming edge, we can continue this traversal for an infinite number of iterations. Since there are a finite number of nodes in G, there must be a point in which a node appears twice during this path construction. For if we travel backwards $n + 1$ times, there are only n nodes, so by the pigeonhole principle, one of the nodes appears twice during this traversal. A node appearing twice in this path is evidence of a cycle. Hence, we have proved by contradiction that there must always be a node without incoming edges in a DAG task graph.

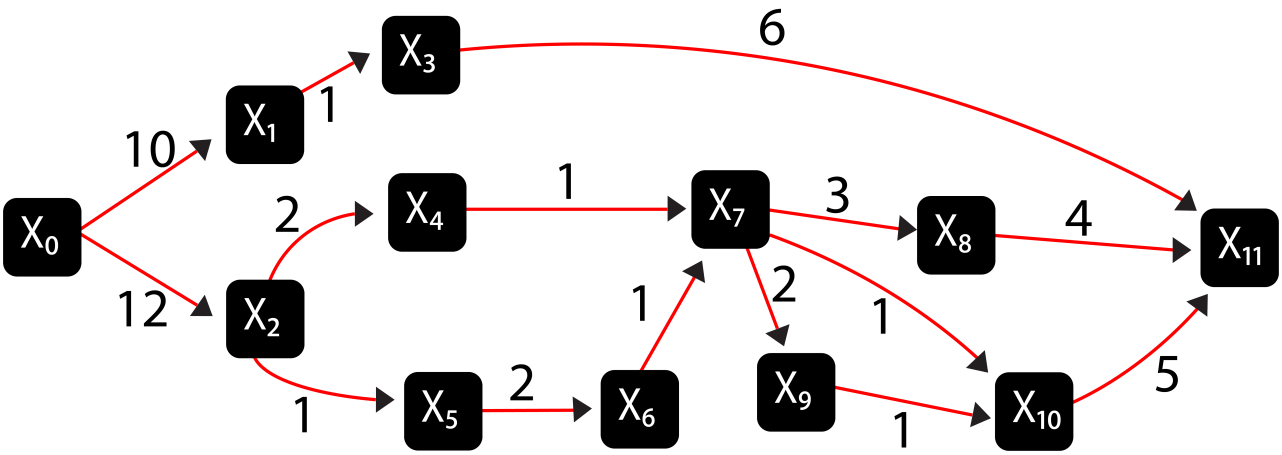
□

Hence, we may assign the index 0 to writing, such that writing is represented by the node X_0. Since 0 is the least nonnegative integer, we see that there can be no other event

with a higher index such that X_0 depends on it. We proceed by temporarily deleting X_0 and edges incident to it. We then repeat the process of finding a node without any incoming edges. This node must exist, either as a node that previously shared an edge with X_0, or as a node that previously had no incoming edges, but was not selected to be the event with the primary index because of an arbitrary tiebreaker. By repeating this process until our task graph has been depleted entirely and breaking ties arbitrarily, we will have produced a topological ordering of our tasks. The topological ordering of our tasks that we will use is the following:

1. X_0 = "Writing"
2. X_1 = "Casting"
3. X_2 = "Crew Building"
4. X_3 = "Marketing"
5. X_4 = "Rehearsal"
6. X_5 = "Scouting Locations"
7. X_6 = "Set Building"
8. X_7 = "Filming"
9. X_8 = "Touring"
10. X_9 = "Visual Effects"
11. X_{10} = "Editing"
12. X_{11} = "Premiering"

For convenience, we see this new notation in the following diagram:



Calculating Earliest Start Times

The earliest start time t_j of a task X_j is the earliest possible time it may commence because all tasks that precede it (all $a \in A = \{X_i \in V \text{ s.t. } (X_i, X_j) \in E\}$), if any, have been running long enough for X_j to begin. To find such a time, we want to know how long it will take for everything that X_j depends on directly to start, (i.e. the earliest start time of each $a \in A$) in addition to for how long these preceding tasks must run before X_j may begin. In our graph, for instance, "filming" (X_7) depends immediately on "rehearsal" (X_4) and "set building" (X_6). This means that "filming" cannot commence unless both "rehearsal" and "set building" have started and have each run for 1 month. Hence, in calculating the earliest start time for "filming", we want to know which sum is larger, the earliest start time of X_4 added to how long it must run before X_7 can start, or the corresponding sum for X_6. This is done using a max function. More concretely, we have

$$t_j = \max\{t_i + w(X_i, X_j)\},$$

where t_j is the desired earliest start time for our task X_j, t_i is the earliest start time for a node X_i that points to X_j, and $w(X_i, X_j)$ is the weight of the edge from X_i to X_j, or for us, the amount of time X_i must run before X_j can begin. Furthermore, we let the start time of the source task be $t_0 = 0$, because it depends on no other task to begin.

It is important to note that for calculating t_j for task X_j, the max function takes as many values as there are incoming edges to X_j. For instance, in calculating t_7, we must consider X_7's incoming edges from X_4 and X_6. Whereas in calculating t_2, we must consider one incoming edge from X_0. Hence the max function takes a variable number of inputs.

Finding earliest starting times is an iterative process in that we require knowledge about the earliest start times of all the nodes that precede a certain node. The given formula reflects the dependence relation on the nodes. We will leave it up to the reader to locate a proof on the correctness of this method for calculating earliest start times, however we suggest inducting on the index n of the tasks, which are already in a topological ordering.

t_j	=	$\max\{t_i + w(X_i, X_j)\}$	
t_0	=	0	0
t_1	=	$\max\{t_0 + w(X_0, X_1)\} = \max\{0 + 10\}$	10
t_2	=	$\max\{t_0 + w(X_0, X_2)\} = \max\{0 + 12\}$	12
t_3	=	$\max\{t_1 + w(X_1, X_3)\} = \max\{10 + 1\}$	11
t_4	=	$\max\{t_1 + w(X_1, X_4), t_2 + w(X_2, X_4)\} = \max\{10 + 3, 12 + 2\}$	14
t_5	=	$\max\{t_2 + w(X_2, X_5)\} = \max\{12 + 1\}$	13
t_6	=	$\max\{t_5 + w(X_5, X_6)\} = \max\{13 + 2\}$	15
t_7	=	$\max\{t_4 + w(X_4, X_7), t_6 + w(X_6, X_7)\} = \max\{14 + 1, 15 + 1\}$	16
t_8	=	$\max\{t_7 + w(X_7, X_8)\} = \max\{16 + 3\}$	19
t_9	=	$\max\{t_7 + w(X_7, X_9)\} = \max\{16 + 2\}$	18
t_{10}	=	$\max\{t_9 + w(X_9, X_{10})\} = \max\{18 + 1\}$	19
t_{11}	=	$\max\{t_{10} + w(X_{10}, X_{11}), t_8 + w(X_8, X_{11}), t_3 + w(X_3, X_1 1)\} = \max\{19 + 5, 19 + 4, 11 + 6\}$	24

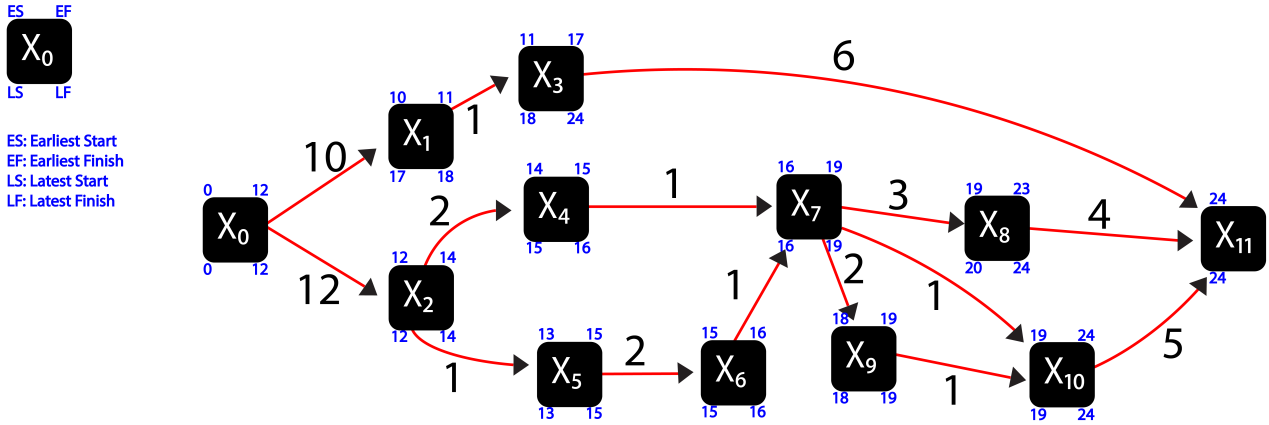
Calculating Latest Start Times

After finding the *earliest start times* we can now calculate the *latest start times*. Similar to that of the earliest, the latest start time for each task is the latest time an event can start without delaying the overall project in any way. For example, looking at X_7, we can see that there are two different paths in which to get there: one taking 15 months and one taking 16. We would take the 16 months as the latest possible time that event could start, because if it starts at 17 months, we would have to delay the project by one month total. The way we calculate these latest start times is by the formula, $T_i = \min\{T_j - w(X_i, X_j)\}$, where T_i is the latest start time (similar to before in which t_i was the earliest start time) and w is the weight of any given edge connecting two tasks of relevance. However, now instead of starting at the beginning of the graph, we start at the end with the final task that is completed and work our way back. So rather than starting at X_0 we start with X_{11}. Since X_{11} is the end of the directed graph, and thus has no edges w leading out of it, we can treat its earliest start time as the same as its latest start time, so $T_n = t_n = 24$. We then work our way backwards, now looking at all the edges directed outwards from X_{10}, which in this case is only one, $w(X_{10}, X_{11})$. So now, inputting this back into our formula we get

$$T_{10} = \min\{T_j - w(X_i, X_j) = \min\{T_{11} - w(X_{10}, X_{11})\} = \min\{24 - 5\} = 19$$

This says that because X_{10} takes five months to complete and we must be premiering at X_{11} in 24 months, then the absolute latest time that the editing task can begin is at the 19 month mark. By doing this throughout the entire graph G we can think of all the latest start times for any given task based on the events that come after it. This is basically thinking of the worst case scenario for each task. This is important because in the next section we will begin to look at the slack of the graph which will compare these earliest and latest start times. By comparing the worst case to the best case scenarios we will be able to see which nodes, or tasks, go on a Critical Path.

T_i	$=$	$\min\{T_j - w(X_i, X_j)\}$	
T_{11}	$=$	24	24
T_{10}	$=$	$\min\{T_{11} - w(X_{10}, X_{11})\} = \min\{24 - 5\}$	19
T_9	$=$	$\min\{T_{10} - w(X_9, X_{10})\} = \min\{19 - 1\}$	18
T_8	$=$	$\min\{T_{11} - w(X_8, X_{11})\} = \min\{24 - 4\}$	20
T_7	$=$	$\min\{T_8 - w(X_7, X_8), T_9 - w(X_7, X_9), T_{10} - w(X_7, X_{10})\} = \min\{20 - 3, 18 - 2, 19 - 1\}$	16
T_6	$=$	$\min\{T_7 - w(X_6, X_7)\} = \min\{16 - 1\}$	15
T_5	$=$	$\min\{T_6 - w(X_5, X_6)\} = \min\{15 - 2\}$	13
T_4	$=$	$\min\{T_7 - w(X_4, X_7)\} = \min\{16 - 1\}$	15
T_3	$=$	$\min\{T_{11} - w(X_3, X_{11})\} = \min\{24 - 6\}$	18
T_2	$=$	$\min\{T_4 - w(X_2, X_4), T_5 - w(X_2, X_5)\} = \min\{15 - 2, 13 - 1\}$	12
T_1	$=$	$\min\{T_3 - w(X_1, X_3)\} = \min\{18 - 1\}$	17
T_0	$=$	$\min\{T_1 - w(X_0, X_1), T_2 - w(X_0, X_2)\} = \min\{17 - 10, 12 - 12\}$	0



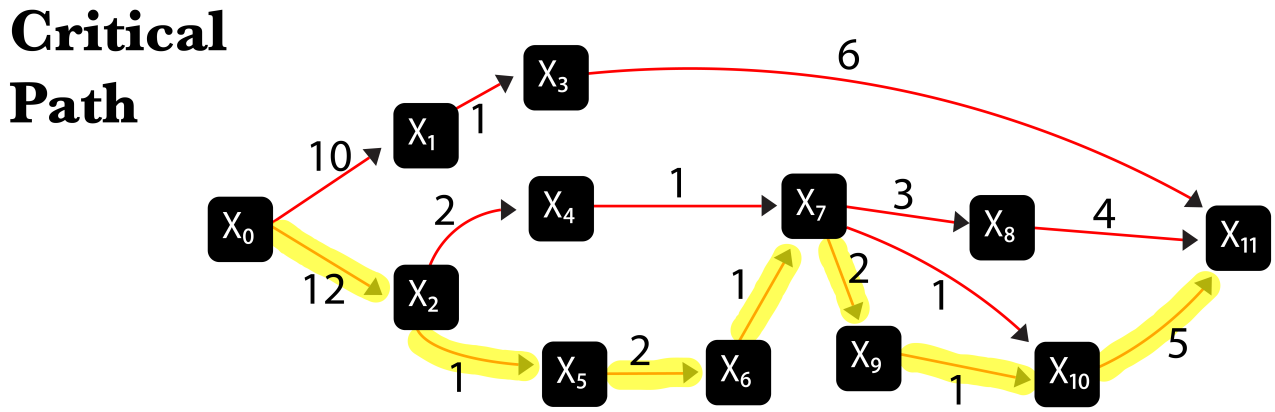
Calculating Slack

After calculating both the earliest and latest start times, we can calculate the slack of the graph. The slack can be thought of as the difference between the latest and earliest start times, or $m_i = T_i - t_i$. This is made easy, as we already have the earliest and latest start times from the tables above. The whole point of calculating slack is to find those vertices, or tasks, that fall on a critical path. If $m_i = 0$, then it falls on a critical path as there is no slack. We look for the vertices that contain zero slack, because by finding the difference between the latest and earliest start times we are actually finding how many months a given task can be delayed before the project as a whole is affected. For example, a task with a slack of one can be delayed by one month without affecting the overall time of the project. But a task with a slack of zero cannot be delayed at all without affecting the overall time of the project, thus that task must be on a critical path. Using the slacks that are calculated below, we can ultimately find the critical path for our film project.

m_i	$=$	$T_i - t_i$	
m_0	$=$	$T_0 - t_0 = 0 - 0$	0
m_1	$=$	$T_1 - t_1 = 17 - 10$	7
m_2	$=$	$T_2 - t_2 = 12 - 12$	0
m_3	$=$	$T_3 - t_3 = 18 - 11$	7
m_4	$=$	$T_4 - t_4 = 15 - 14$	1
m_5	$=$	$T_5 - t_5 = 13 - 13$	0
m_6	$=$	$T_6 - t_6 = 15 - 15$	0
m_7	$=$	$T_7 - t_7 = 16 - 16$	0
m_8	$=$	$T_8 - t_8 = 20 - 19$	1
m_9	$=$	$T_9 - t_9 = 18 - 18$	0
m_{10}	$=$	$T_{10} - t_{10} = 19 - 19$	0
m_{11}	$=$	$T_{11} - t_{11} = 24 - 24$	0

Finding Critical Paths

Since $X_0, X_2, X_5, X_6, X_7, X_9, X_{10}$, and X_{11} have corresponding slack values of 0, these tasks are on one or more *critical paths*. Coincidentally, these nodes are all on the same path, so it turns out our project has *one* critical path.



This means that Henry and William will devote more energy and resources to starting these highlighted tasks on time as they must proceed smoothly if the movie is to come out on schedule. Although our project has one critical path, it is easy to illustrate a scenario in this movie project where another critical path arises. [1]
To recap, we will put our greatest efforts into ensuring that

1. writing
2. crew building
3. scouting locations
4. set building
5. filming
6. visual effects
7. editing
8. and premiering

happen on time, so that audiences can enjoy our movie 24 months from now, in 2023.

[1] Suppose touring must happen for 5 months, instead of 4 months, before premiering can happen. Then the latest start time for the node X_8 will decrease by 1 month, which decreases m_8 to 0. Thus, in addition to the critical path in the original design, we would also include the following critical path: $\{X_0, X_2, X_5, X_6, X_7, X_8, X_{11}\}$.

Work Cited

Chatterjee, Upasana. “Applications of Three Common Graph Theoretical Algorithms.” Research Academy for Young Scientists, 10 July 2014, doi:March 20th, 2021.