Critical Path Method

Implementations & Analysis

Yuqi Hu
Li Hao
Hejia Zhang
(last name alphabetic order)

What is Critical Path Method?

- CPM is a step-by-step project management technique to identify activities on the critical path.

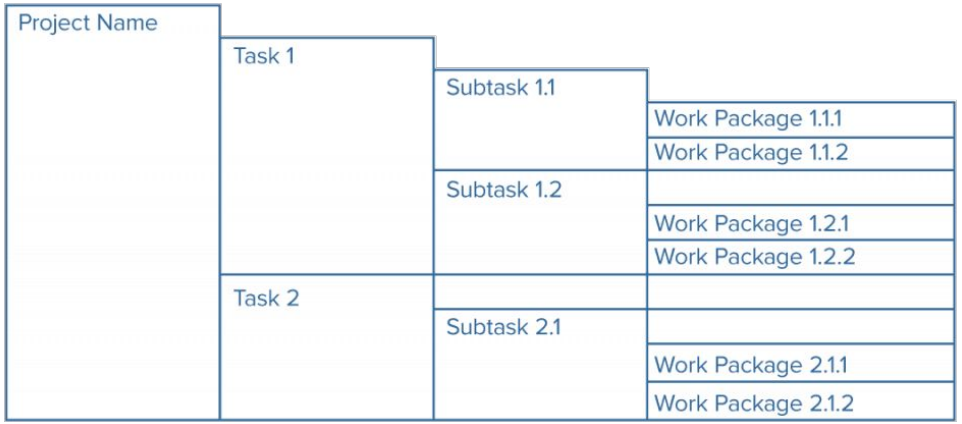
- It was first introduced in the back in the late 1950s and was used in the Manhattan project.

- It shows activities as nodes, usually represented by boxes, and use arrows to show logical links between them.

- A critical path is determined by identifying the longest stretch of dependent activities and measuring the time required to complete them from start to finish.

What is Critical Path Method?

The CPM begins by breaking down the project into individual tasks or activities. Each task is assigned an estimated duration, which is then used to calculate the earliest possible start and finish dates for each activity. The CPM then identifies the longest path of activities from start to finish, known as the critical path. This path represents the minimum amount of time required to complete the project.



Why use it?

The critical path is the longest path in the project network diagram:

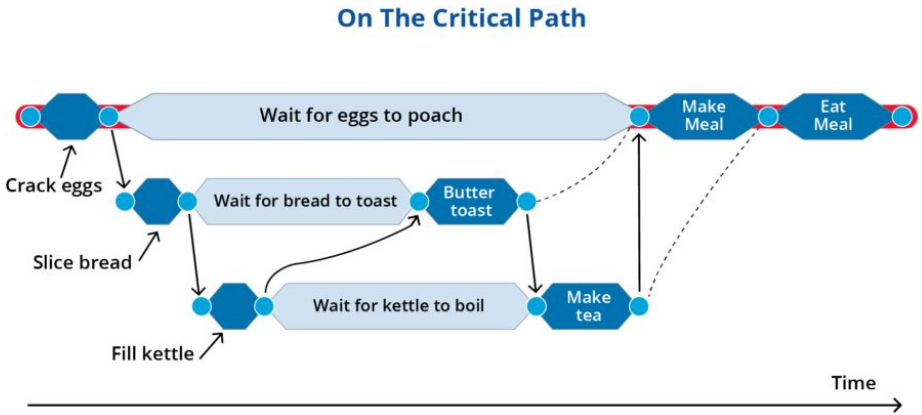
- Any activity on the critical path, if delayed, can delay the project;
- A critical path also gives the shortest time possible to complete the project;
- It is possible to have more than one critical path on a project.

The “critical path” is the longest path in the network with only zero float activities

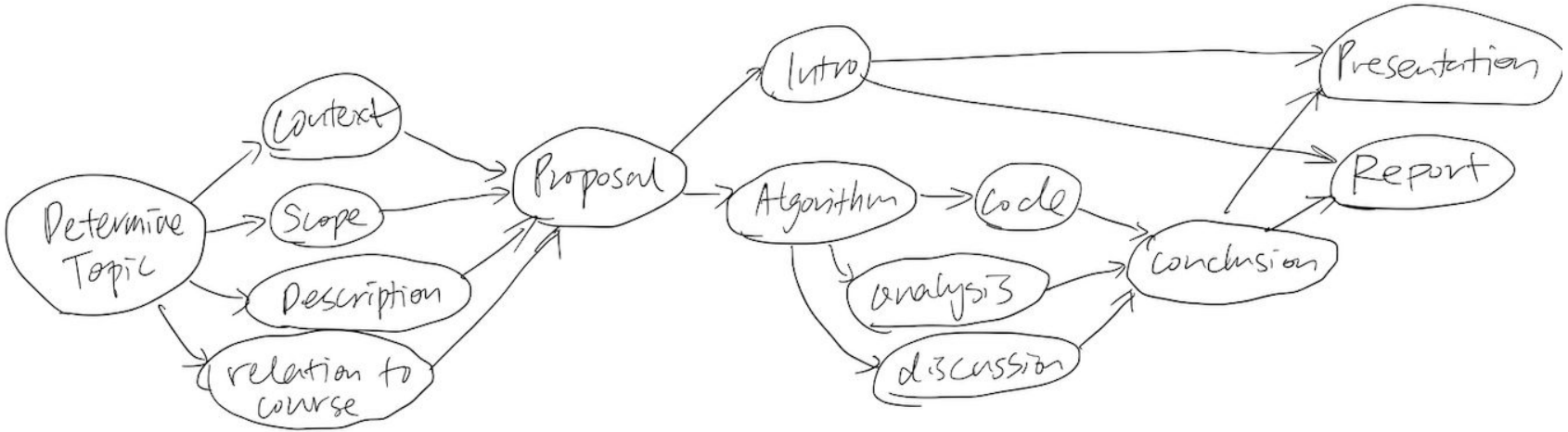
Why do we need the critical path?

The critical path is the longest path in the project network diagram:

- Prioritize tasks: simplify essential and non essential tasks in a visual way
- Prevente delays: allocate time required to complete the project and keep track of the longest sequence
- Enhance productivity: reschedule low-priority tasks
- Track and manage task efficiently: make modifications to the project and track the progress in real-time



Why do we need the critical path?



Components

Earliest start time - earliest possible date you can start an activity considering the dependencies.

Earliest finish time - earliest possible date you can complete an activity considering its ES and duration.

Latest start time - last possible date you can start an activity before causing a significant project delay

Latest finish time - latest possible date you can complete a task based on its LF and duration.

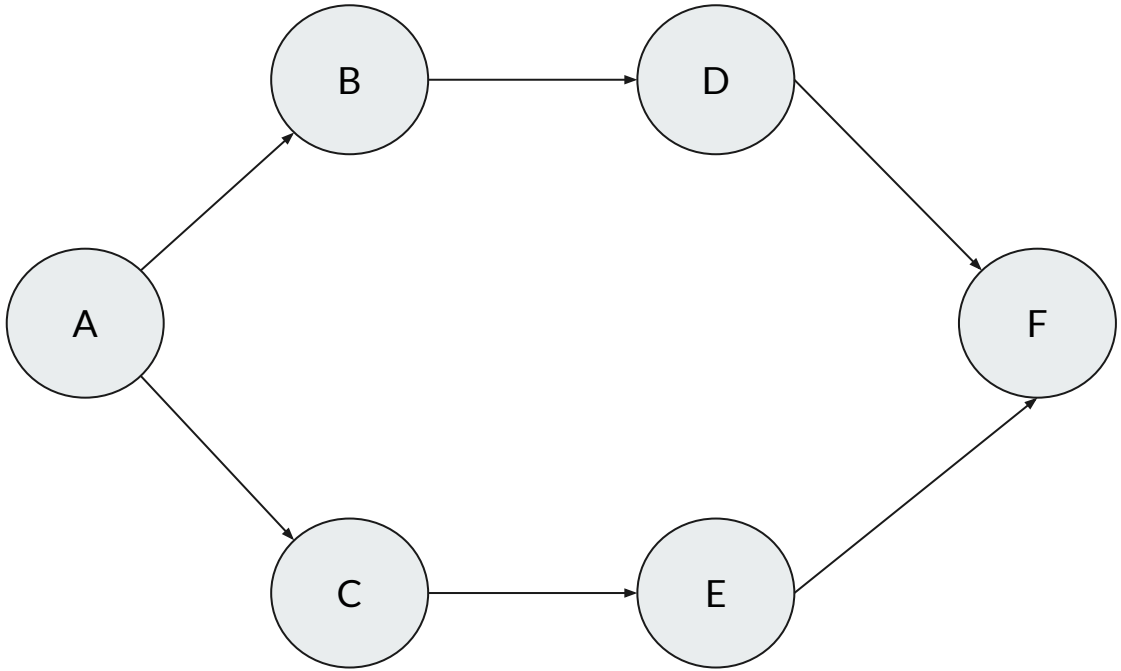
Task duration (t) - total amount of time it takes to complete an activity.

What is Critical Path Method?

Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D, E	4

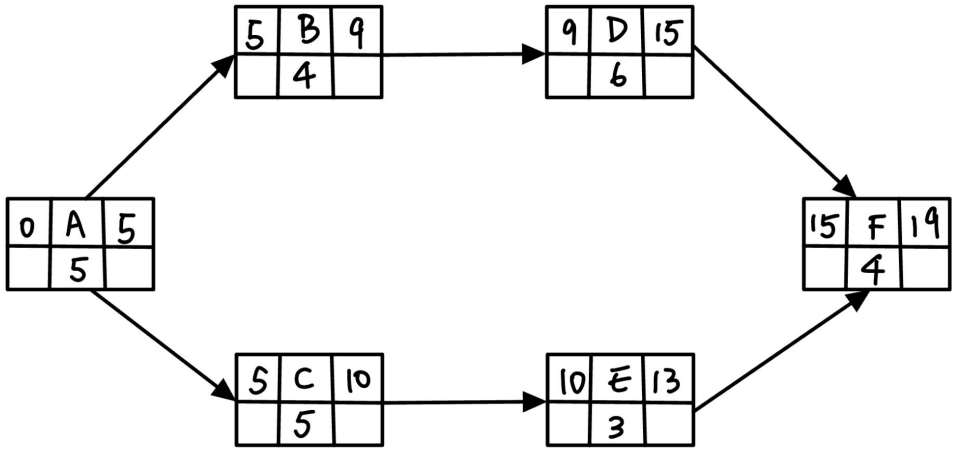
Step 1 - Generate a project network diagram

Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D, E	4



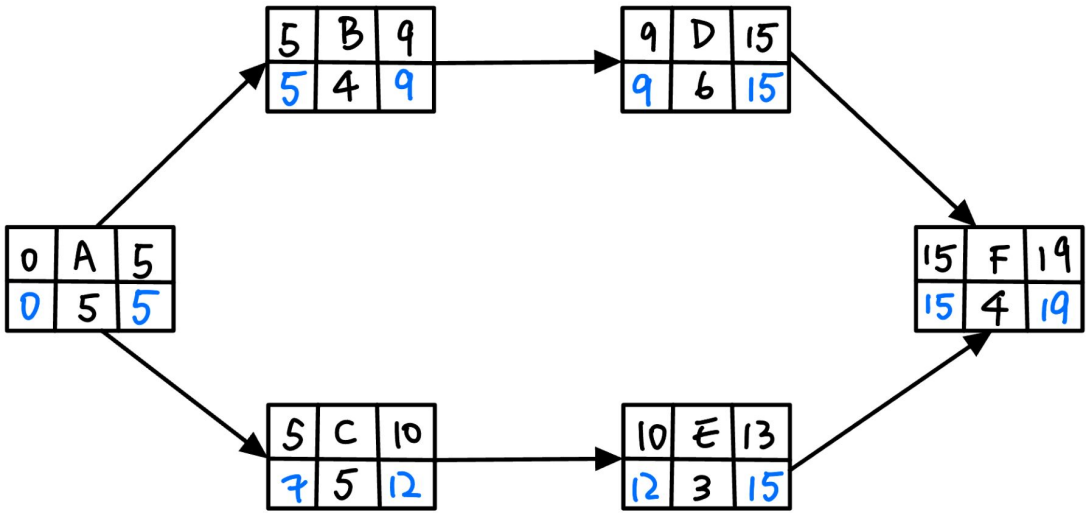
Step 2 Determine ES and EF

Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D, E	4



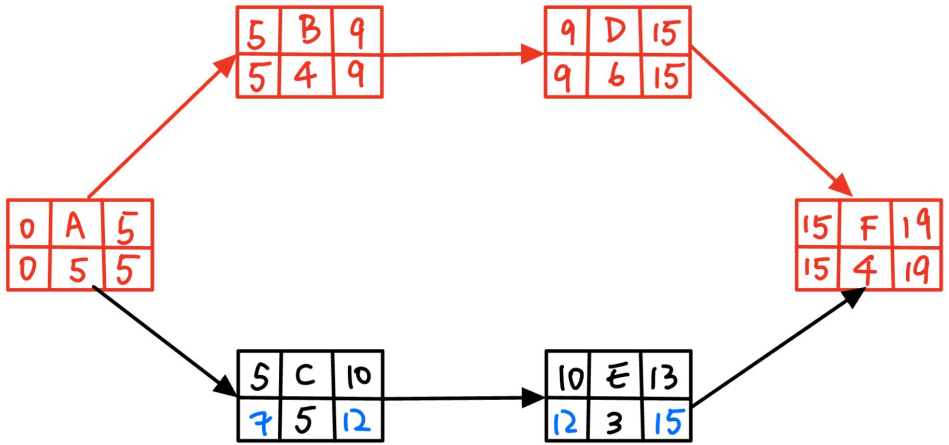
Step 2 Determine LS and LF

Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D, E	4



Step 2 Find the critical path

Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D, E	4

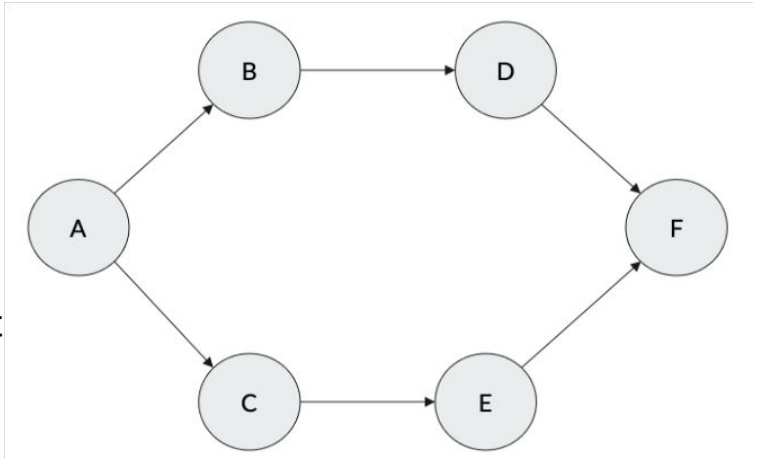


PseudocodeCode

Define a **Task structure** with fields for:
- Task ID (task_id)
- Duration (duration)
- Earliest Start Time (earliest_start)
- Earliest Finish Time (earliest_finish)
- Latest Start Time (latest_start)
- Latest Finish Time (latest_finish)
- Slack Time (slack)

Define a **Graph structure** to represent the project, with nodes and edges:
- Nodes represent tasks (Task objects)
- Edges represent dependencies between tasks

PseudocodeCode



Initialize an empty **graph G** to represent the project

Create a function **ConstructGraph(tasks):**

a. Create an empty graph G.

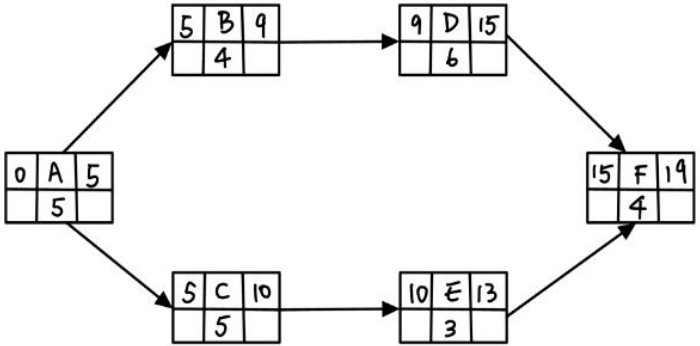
b. For each task in tasks:

i. Add a node to G representing the task with task_id as node ID and task object as node attributes.

ii. For each dependency in task.get('dependencies', []):

- Add a directed edge from dependency task to the current task in G.

Pseudocode Code



Create a function **CalculateEarliestTimes(tasks, G)**:

a. For each task in tasks:

i. Add a node to G representing the task.

ii. Set the node's duration based on task.duration.

iii. If task has no dependencies:

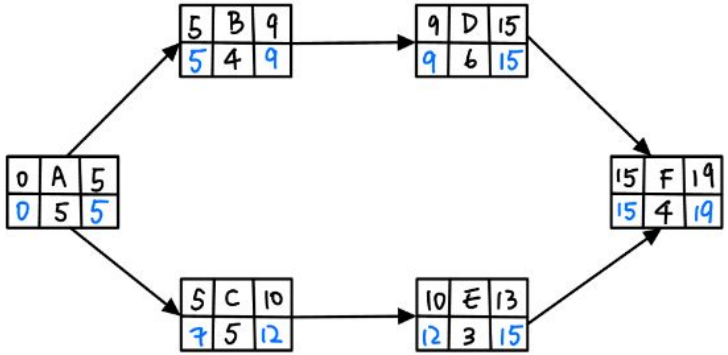
- Set task's earliest_start to 0 and earliest_finish to duration.

iv. Else:

- Set task's earliest_start to maximum earliest_finish of dependencies.

- Set task's earliest_finish to earliest_start + duration.

Pseudocode Code



Create a function **CalculateLatestTimes(tasks, G, project_duration)**:

a. Set project_duration to the maximum earliest_finish of all tasks.

b. For each task in tasks in reverse topological order of G:

i. If task has no successors:

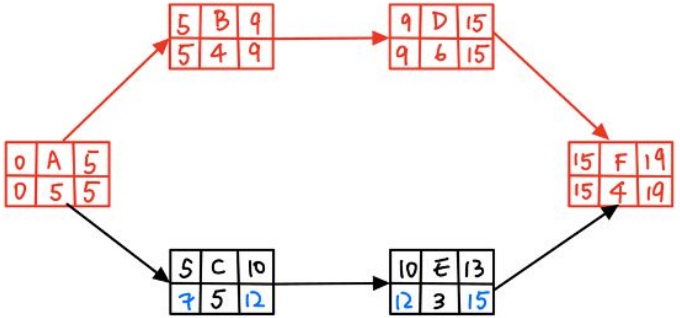
- Set task's latest_finish to project_duration.

ii. Else:

- Set task's latest_finish to minimum latest_start of successors.

- Set task's latest_start to latest_finish - duration.

Pseudocode Code



Create a function **CalculateSlack(tasks, G)**:

a. For each task in tasks:

i. Calculate task's slack as latest_start - earliest_start.

Create a function **IdentifyCriticalPath(tasks)**:

a. Initialize an empty list critical_path.

b. For each task in tasks:

i. If task's slack is 0, add task to critical_path.

Time & Space Complexity

- Time Complexity: O(V + E) for constructing the graph and performing critical path calculations(forward + backward passes).

- Space Complexity: O(V + E) for storing the graph and intermediate calculations.

Alternative Method - Brute Force (Backtracking)

To find it, we can convert the tasks in to a weighed, directed graph.

In the Brute Force method, we can simply use backtracking to explore all possible paths in the graph and find the critical path among them.

In the worst case, there are $O(2^N)$ to be explore and O(N) edges on each path. So the time complexity of this method is $O(N * 2^N)$ where N is the number of tasks.

Alternative Method - Dijkstra's

- Dijkstra's find the shortest path, not longest path. So we can reverse all the weights (w = -w), so the the "shortest path" that Dijkstra finds is actually longest. Since all weights are negative now and there are no cycles, Dijkstra's does work here.
- The time complexity is O(V * (V+E)logV)

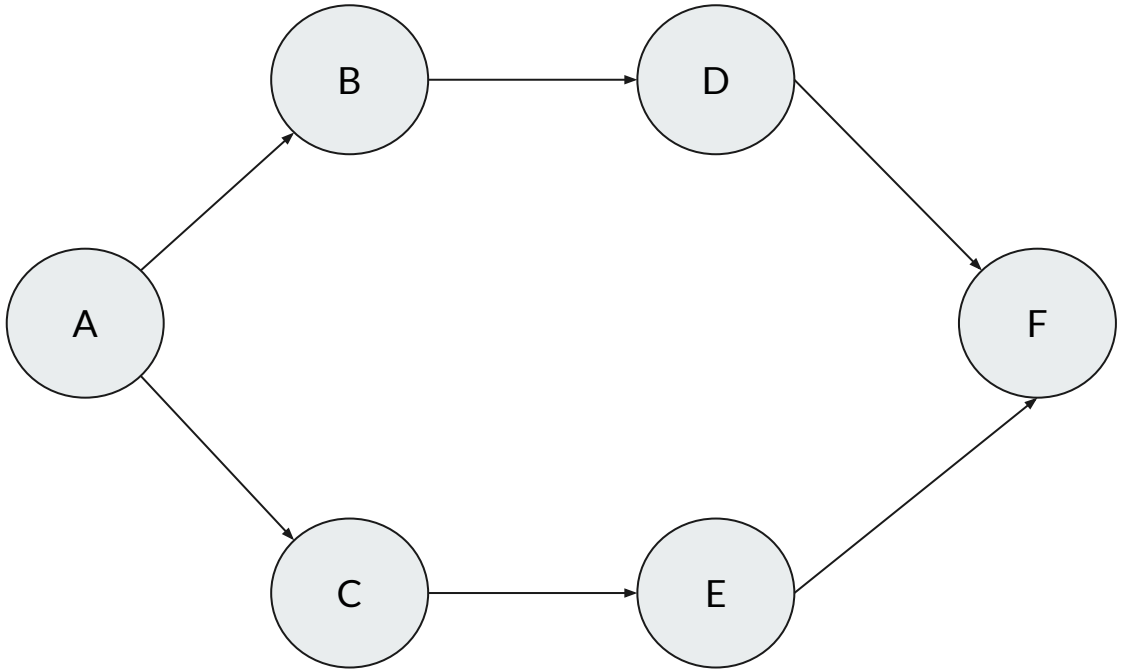
pseudocode:

- For each node:
    Use Dijkstra's to find the longest path to every reachable node
    Keep record of that path

  Find the longest path among the candidates

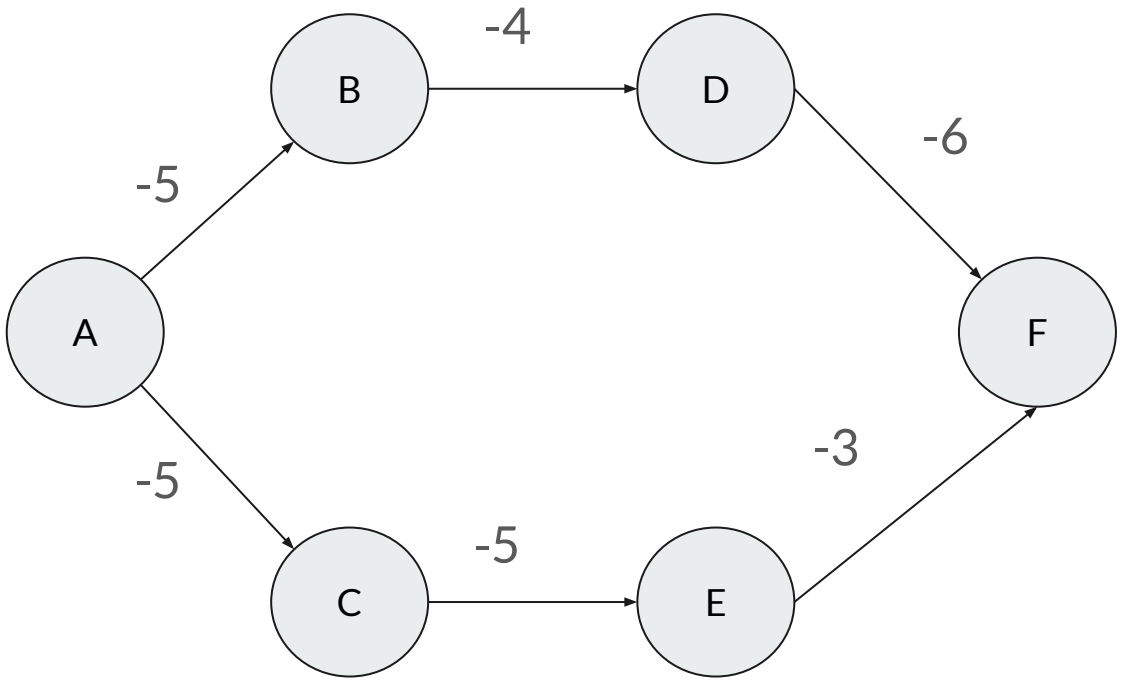
Dijkstra's Illustration

Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D, E	4



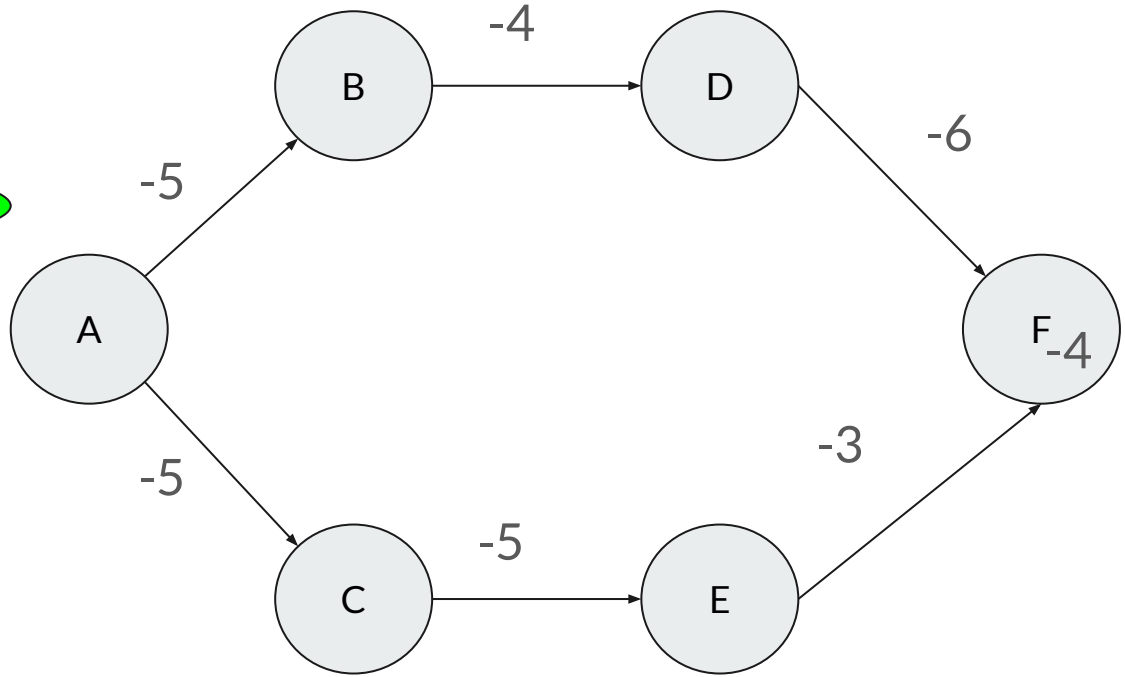
Dijkstra's Illustration

Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D, E	4



Dijkstra's Illustration

Start	"Shortest path"	Total weight
A	ABDF	-19
B	BDF	-14
C	CEF	-2
D	DF	-10
E	EF	-7
F	F	-4



Performance Comparison

	Test Case Size 1-50	Test Case Size 50-100
Critical Path Method	0.000476460	0.00157213
Brute Force	14.8626	Too slow, probably takes days
Dijkstra’s	0.000765674	0.00728725

Limitations

By implementing different algorithm for CPM, we gain a clear insight for this topic. However, CPM itself has its own weaknesses as well.

1. Overlooks non-critical tasks - this method can help us focus on critical tasks but it can also cause delays in completion of your projects
2. Time-consuming - designing a CPM chart is time consuming, especially for larger projects, you need to go through hundreds of tasks and dependency relationships
3. Not realistic IRL - you have to predetermine the task durations, but IRL errors and delays happen, so the timeline is not always reliable

Comparison wrap up

- CPM can be implemented using brute force but only recommended when project is small
- For large projects in real world scenario, we should either be using **critical path method** or **dijkstra's method** approach for the purpose of efficiency
- You should evaluate your projects beforehand and see whether CPM is suitable for you.

Reference

- https://cloudfresh.com/en/blog/critical-path-method-how-it-influences-the-project-management-process/
- https://blog.hubspot.com/marketing/critical-path-method
- https://networkx.org/documentation/stable/reference/introduction.html