Introduction to Mathematical Modeling

Ramapo College of New Jersey
Instructor: Dr. Atul Anurag

Semester: Fall 2025 Date: September 18, 2025

Introduction

Graphs are fundamental structures in computer science and mathematics, used to model pairwise relations between objects. A **tree** is a special type of graph with unique properties. Closely related are the concepts of **spanning trees** and **minimal spanning trees**, which have important applications in network design, optimization, and algorithms.

Tree

Definition

A **tree** is an undirected graph that is:

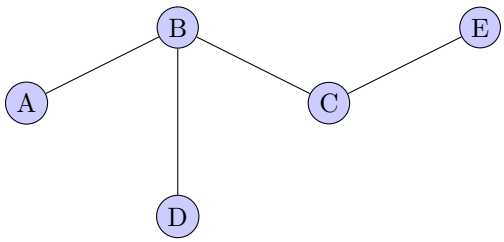
- Connected: There is a path between any two vertices.
- Acyclic: It contains no cycles.

Equivalently, a tree with n vertices has exactly $n - 1$ edges.

Properties

- There is a unique path between any two vertices.
- Adding any edge creates exactly one cycle.
- Removing any edge disconnects the graph.

Example of a Tree



Spanning Tree

Definition

Given a connected undirected graph $G = (V, E)$, a **spanning tree** is a subgraph $T = (V, E_T)$ that:

- Includes all vertices V of G.
- Is a tree (connected and acyclic).

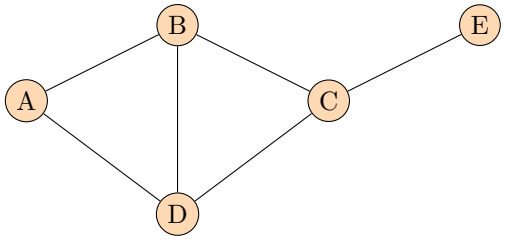
Note that a graph can have multiple spanning trees.

Properties

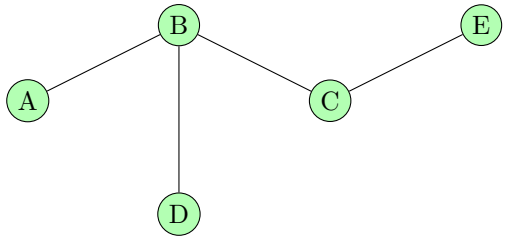
- A spanning tree of a graph with n vertices has exactly $n - 1$ edges.
- It connects all vertices with minimal number of edges to maintain connectivity.

Example of a Graph and Its Spanning Tree

Graph:



A Spanning Tree:



Minimal Spanning Tree (MST)

Definition

A **Minimal Spanning Tree** of a connected weighted graph is a special kind of spanning tree that connects all the vertices together with the smallest possible total edge weight.

In other words, among all the spanning trees of the graph, the MST is the one where the sum of the weights of its edges is the least.

Example: If you think of the graph as a network of cities connected by roads with different lengths, the MST represents the cheapest way to connect all cities without any loops.

Properties

- MST connects all vertices with minimum possible total edge weight.
- MST is not necessarily unique if multiple edges have the same weights.

Algorithms to find MST

- Kruskal's Algorithm

Kruskal's Algorithm

Overview: Kruskal's algorithm is a greedy method to find the Minimal Spanning Tree (MST) of a connected weighted graph. The key idea is to repeatedly add the next lightest edge that does not form a cycle until all vertices are connected.

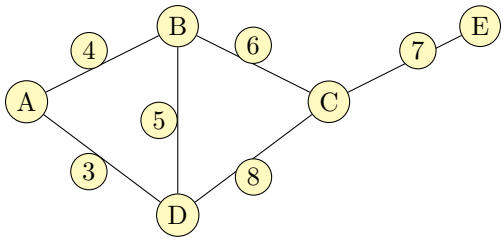
Algorithm Steps:

1. Sort all the edges of the graph in non-decreasing order of their weights.
2. Initialize the MST as an empty set.
3. Iterate over the sorted edges and for each edge:
    - Check if adding the edge to the MST will create a cycle.
    - If it does not create a cycle, add the edge to the MST.

4. Stop when the MST contains $n-1$ edges, where n is the number of vertices.

Cycle Detection: To efficiently detect cycles, Kruskals algorithm uses a *Disjoint Set Union* (DSU) or *Union-Find* data structure that keeps track of which vertices are in which components.

Example:
Consider the weighted graph:



Sorting edges by weight:

$$(A-D:3),(A-B:4),(B-D:5),(B-C:6),(C-E:7),(C-D:8)$$

- Add $A-D$ (weight 3) no cycle.
- Add $A-B$ (weight 4) no cycle.
- Add $B-D$ (weight 5) adding this creates a cycle $A-B-D-A$, so skip it.
- Add $B-C$ (weight 6) no cycle.
- Add $C-E$ (weight 7) no cycle.

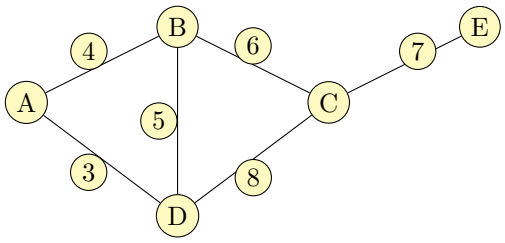
Edges in MST:

$$\{(A-D),(A-B),(B-C),(C-E)\}$$

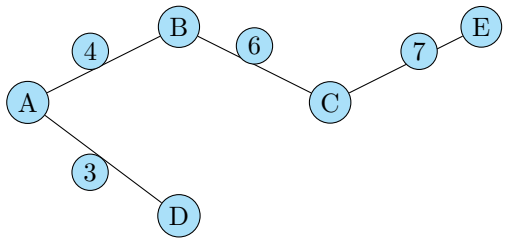
Total weight $= 3+4+6+7 = 20$.

Example of a Weighted Graph and its MST

Weighted Graph:



Minimal Spanning Tree:



Note: This example MST includes edges with total weight $3+4+6+7=20$. Depending on the algorithm, you might get a different MST with same minimal weight.

1 Order Requirement Digraph

1.1 Definition

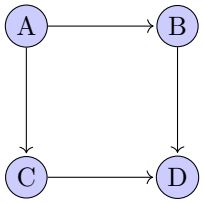
An **Order Requirement Digraph** (also called a *precedence graph* or *task dependency graph*) is a directed graph used to represent tasks and their order of execution.

- Each vertex represents a task or activity.
- A directed edge from vertex A to vertex B (written $A \to B$) means task A must be completed before task B can start.

This type of graph helps model dependencies between tasks in projects or processes.

1.2 Properties

- The graph must be **acyclic** (no cycles) because a cycle would mean tasks depend on each other in a loop, which is impossible.
- It is often called a **Directed Acyclic Graph (DAG)**.



2 Critical Path Analysis (CPA)

2.1 Introduction

Critical Path Analysis is a project management technique used to identify the longest sequence of dependent tasks (called the **critical path**) that determines the shortest possible time to complete the entire project.

2.2 Key Concepts

- **Tasks/Activities**: Basic units of work, with a known duration.
- **Dependencies**: Which tasks must be finished before others start (modeled by the order requirement digraph).
- **Critical Path**: The longest path through the dependency graph it determines the minimum project completion time.
- **Slack Time**: The amount of time a task can be delayed without affecting the overall project duration.

2.3 Steps to Perform CPA

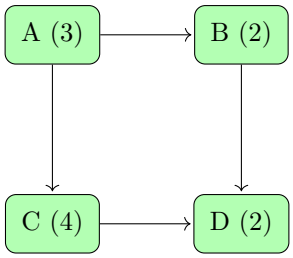
1. Draw the order requirement digraph showing all tasks and dependencies.
2. Assign a duration to each task.
3. Calculate the **Earliest Start Time (EST)** and **Earliest Finish Time (EFT)** for each task, moving forward through the graph.
4. Calculate the **Latest Start Time (LST)** and **Latest Finish Time (LFT)** for each task, moving backward from the project's end.
5. Identify tasks with zero slack (i.e., where EST = LST) these form the critical path.

2.4 Example

Consider the following tasks with durations and dependencies:

Task	Duration (days)	Depends on
A	3	–
B	2	A
C	4	A
D	2	B, C

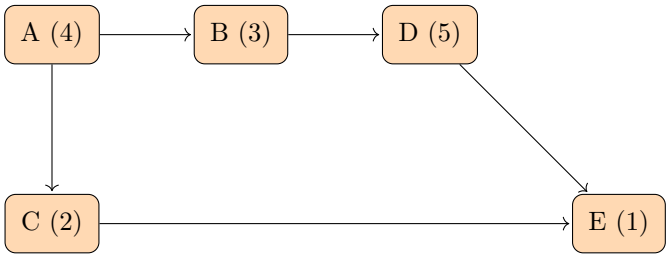
Dependency graph:



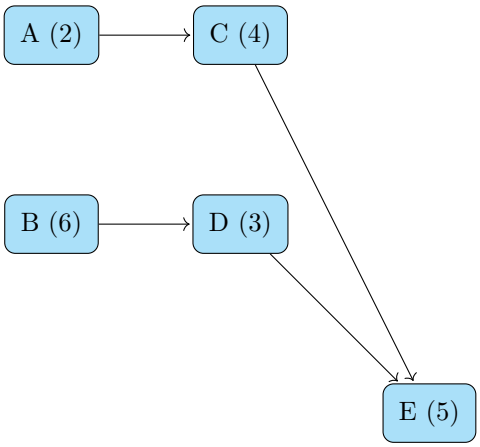
Example 2: Critical Path Calculation

- Path 1: $A \to B \to D$ with total duration $3 + 2 + 2 = 7$ days.
- Path 2: $A \to C \to D$ with total duration $3 + 4 + 2 = 9$ days.

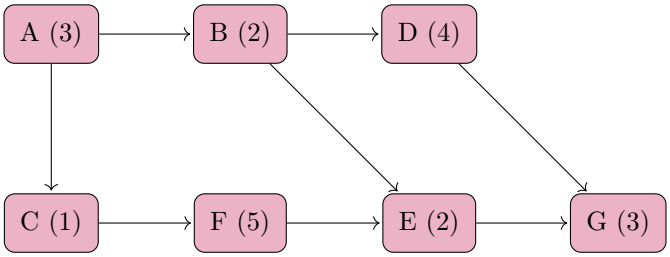
The critical path is $A \to C \to D$, since it takes the longest time (9 days). This means the project cannot finish in less than 9 days, and any delay in tasks A, C, or D will delay the entire project.



Example 3



Example 3



End of Lecture #5