Introduction to Graphs, Trees, DFS, BFS, and Dijkstra

Opening

- Google Maps showing routes between cities.
- Instagram / social media showing friends and followers.
- Timetable or course prerequisites.

Key Idea

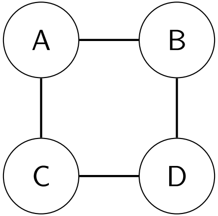
All of these can be represented using one idea: **graphs**.

What is a Graph?

Definition

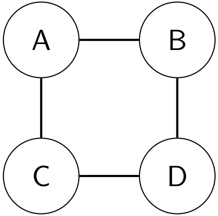
A **graph** consists of:

- A set of **nodes** (or **vertices**)
- A set of **edges** (or **links**) connecting pairs of nodes



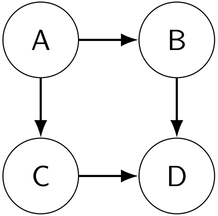
Types of Graphs

Undirected Graph



- Edges have no direction.
- Example: Two-way roads.

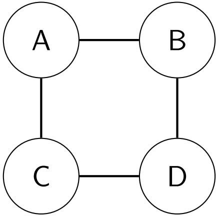
Directed Graph



- Edges have direction (arrows).
- Example: “follows” on Twitter.

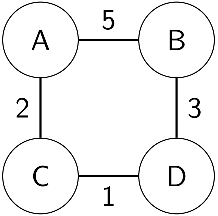
Weighted vs Unweighted Graphs

Unweighted



- All edges treated as cost $= 1$.

Weighted



- Each edge has a cost (distance, time, money).

Why Study Graphs?

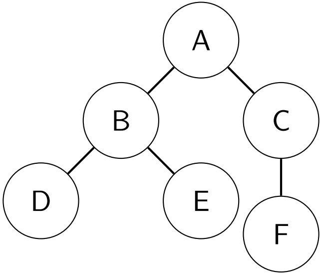
- Google Maps: shortest paths between locations.
- Social networks: connections between people.
- File systems: folders and files.
- Power or communication networks.
- Dependencies in software and compilers.
- AI pathfinding in games.

What is a Tree?

Definition

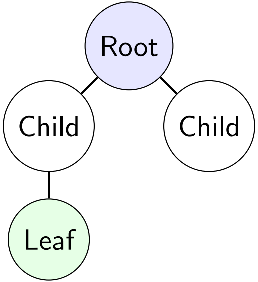
A **tree** is a special graph that:

- Has no cycles.
- Has exactly one simple path between any two nodes.
- If it has N nodes, it has $N - 1$ edges.



Tree Terminology

- **Root**: The chosen starting node (A in the diagram).
- **Parent / Child**: B is a child of A; A is parent of B.
- **Leaf**: Nodes with no children (D, E, F).
- **Depth**: Distance (in edges) from the root.



Uses of Trees

- File/folder structure in operating systems.
- Binary search trees for fast lookup.
- Organizational charts.
- Minimum spanning trees in networks.

Why We Need Traversal

Main Question

Given a graph, how do we:

- Visit all nodes?
- Check if the graph is connected?
- Search for a particular node?

- Two fundamental graph traversal methods:
  - **Depth First Search (DFS)**
  - **Breadth First Search (BFS)**

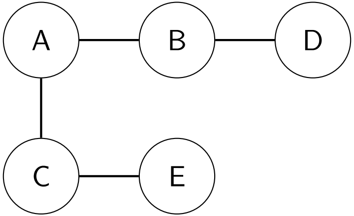
DFS Intuition

Idea

Go as deep as possible before backtracking.

- Like exploring a maze by always taking a new corridor until you get stuck, then backtrack.
- Can be implemented with recursion or with an explicit stack.

DFS Example Graph



Example DFS starting at A:

$$A \to B \to D \to C \to E$$

(Order can vary slightly depending on neighbor order.)

DFS Procedure (Conceptual)

1. Start at a given node (source).
2. Mark it as visited.
3. For each unvisited neighbor:
   - Perform DFS on that neighbor.
4. When no unvisited neighbors remain, backtrack.

Uses of DFS

- Exploring all possible paths.
- Detecting cycles.
- Topological sorting.
- Solving puzzles (e.g., mazes, Sudoku).

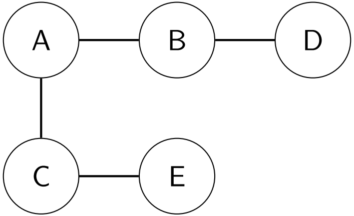
BFS Intuition

Idea

Explore level by level, like ripples spreading out from a stone in water.

- Uses a **queue**.
- All nodes at distance 1, then distance 2, etc.

BFS Example Graph



Example BFS starting at A:

$$A \to B \to C \to D \to E$$

(First visit all neighbors of A, then neighbors of those, and so on.)

BFS Procedure (Conceptual)

1. Start at source, mark as visited, and push into a queue.
2. While queue is not empty:
   - Pop a node from the front.
   - For each unvisited neighbor:
     - Mark it visited.
     - Push it into the queue.

Uses of BFS

- **Shortest path in unweighted graphs**.
- Finding connected components.
- Level order traversal of trees.

BFS vs DFS

Feature	BFS	DFS
Data structure	Queue	Stack / Recursion
Exploration style	Level-by-level	Go deep first
Shortest path (unweighted)	**Yes**	**No (not guaranteed)**
Typical uses	Shortest paths, levels	Cycles, full search, puzzles

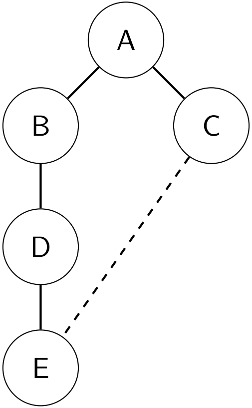
DFS and Shortest Paths

Key Idea

DFS chooses one direction and goes as deep as possible, without checking if there is a **shorter alternative path**.

- It may find a path to the target.
- But that path might not be the shortest one.

Example: DFS vs Shortest Path



- Shortest path from A to E is: $A \rightarrow C \rightarrow E$ (2 edges).
- DFS might go: $A \rightarrow B \rightarrow D \rightarrow E$ (3 edges).
- DFS gets "distracted" by going deep on one side.

Why BFS Finds the Shortest Path

- BFS explores in **layers**:
  - Distance 0: source node.
  - Distance 1: neighbors.
  - Distance 2: neighbors of neighbors.
- The first time BFS reaches the target, it has used the minimum number of edges.

Why BFS is Not Enough

- BFS assumes all edges have the same cost (e.g., cost $= 1$).
- Real life: roads have different lengths, speeds, or tolls.
- We need an algorithm for **shortest path in weighted graphs**.

Solution

Dijkstra's Algorithm

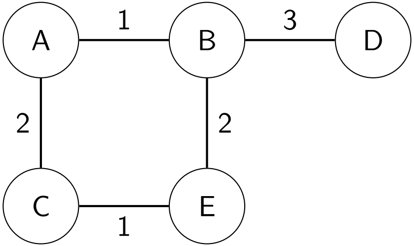
Dijkstra: Intuition

Idea

Always extend the path that currently has the **smallest total cost**.

- Start with distance 0 at the source.
- Repeatedly pick the closest unvisited node.
- Update (relax) distances to its neighbors.
- This is like BFS but using a **priority queue** instead of a normal queue.

Dijkstra Example



- Shortest path from A to D?
- Work through distances step by step in class.

Dijkstra: Where is it Used?

- GPS / Google Maps routing.
- Network routing protocols (e.g., OSPF).
- Game AI pathfinding.
- Robotics navigation.

Summary

- Graph = nodes + edges (can be directed/undirected, weighted/unweighted).
- Tree = special graph with no cycles and exactly one path between nodes.
- DFS = go deep first (good for full exploration, cycles, puzzles).
- BFS = level-by-level (gives shortest path in unweighted graphs).
- DFS **cannot guarantee** shortest path efficiently.
- Dijkstra = shortest path in weighted graphs.

Quick Class Activity

- Give students a small graph (5–7 nodes).
- Ask them to:
  - Write one possible DFS order.
  - Write one possible BFS order.
  - Find the shortest path between two given nodes.
  - Decide if that graph is a tree or not.

Questions?