Usages of graphs

Algorithm	Main Uses	Graph Type	Tips / Notes
DFS	- Explore graph- Find connected components- Detect cycles	Directed & Undirected	Uses recursion – be careful of stack overflow
BFS	- Shortest path in unweighted graphs- Level order traversal	Directed or Undirected	Distance = number of steps
Dijkstra	- Shortest path (non-negative weights)	Directed or Undirected + no negative edges	Use priority queue (min-heap), avoid revisiting nodes
Bellman-Ford	- Shortest path with negative weights- Detect negative cycles	Mostly Directed	Slower (O(n·m)), but powerful
Floyd-Warshall	- All pairs shortest path- Transitive closure	Directed or Undirected	$O(n^3)$ – best for small graphs (n ≤ 400)
Topological Sort	- Task ordering- Solve DAG problems	Directed Acyclic Graph (DAG) only	If there's a cycle → no topo sort is possible
Union-Find (DSU)	- Check if nodes are in same component- Kruskal's MST	Mostly Undirected	Use path compression + union by size
0-1 BFS	- Shortest path with only weights 0 or 1	Directed or Undirected	Faster than Dijkstra in this case
Path Reconstruction	- Print the actual path, not just the cost	Any	Keep a `parent[]` while doing BFS/Dijkstra
Cycle Detection (DFS)	- Detect cycles- Check if graph is cyclic or acyclic	Directed or Undirected	For directed: cycle if we visit a node in the call stack
Connected Components	- Count components- Group nodes	Mostly Undirected	Use DFS, BFS, or Union-Find

Graph Type	What it Means	Examples / Notes
Unweighted	No edge weights	Mazes, Social networks
Weighted	Each edge has a weight/cost	Roads with distances or costs
Directed	Edges have directions (u → v)	Task dependencies, DAG
Undirected	Edges go both ways (u ↔ v)	Basic road maps, friend connections
DAG	Directed + no cycles	Task scheduling, build order
Dense Graph	Many edges, close to n^2	Use adjacency matrix
Sparse Graph	Few edges	Use adjacency list