⛅

DAGS

📘 Notes: DAGs, Topological Sort, and Shortest Paths

🔷 Directed Acyclic Graphs (DAGs)

- A **DAG** is a directed graph with **no directed cycles**lecture08_dags_toposort.
- Generalization of a tree:
  - Trees = special case (1 root, 1 parent per node).
  - DAGs allow multiple roots, multiple parents, and more edges.
- Useful for modeling **dependencies** (e.g., build systems, course prerequisites).
- Key concepts:
  - **Sources**: nodes with no incoming edges.
  - **Sinks**: nodes with no outgoing edges.

🔷 Topological Sort

Definition: An ordering of vertices where if (u,v)∈E(u, v) \in E(u,v)∈E, then uuu appears before vvv
lecture08_dags_toposort.

- Represents a valid order to process tasks with dependencies.
- A DAG can have **many valid topo orders**.

Algorithms

1. **Kahn's Algorithm**lecture08_dags_toposort:

- Count indegrees.
- Put all indegree=0 nodes into a queue.
- Repeatedly pop, add to order, and decrease indegrees of children.
- Complexity: **O(V+E)**.

2. **DFS-based Toposort**lecture08_dags_toposort:
   - Run DFS, record vertices in **postorder** (finish times).
   - Reverse the postorder → topological order.
   - Complexity: **O(V+E)**.

🔷 Shortest Path in a DAG

Key insight: Toposort gives the perfect order to relax edges.

- In Dijkstra/Bellman-Ford, the challenge is deciding order.
- In DAGs, **toposort ensures parents are processed before children**lecture08_dags_toposort.

Algorithm (gist)

1. Compute topological order of DAG.
2. Initialize distances:
   - d(s)=0d(s) = 0d(s)=0 for source sss.
   - d(v)=∞d(v) = \inftyd(v)=∞ (or `None`) for others.
3. Process vertices in topo order:
   - For each edge (u,v)(u, v)(u,v):d(v)=min(d(v),d(u)+w(u,v))

     d(v)=min(d(v), d(u)+w(u,v))d(v) = \min(d(v), \; d(u) + w(u, v))
   - Since uuu comes before vvv, all shortest paths into uuu are already known.

Complexity

- Toposort = **O(V+E)**.

- Relax edges once each = **O(V+E)**.
- ✅ Total = **O(V+E)** (faster than Dijkstra or Bellman-Ford on general graphs).lecture08_dags_toposort

Implementation gist

```
def dag_shortest_paths(adjlist, s):
    order = toposort([[v for v, _ in vs] for vs in adjlist]) # topo sort
    dists = [None] * len(adjlist)
    dists[s] = 0
    for u in order:
        if dists[u] is None:
            continue
        for v, w in adjlist[u]:  # relax edges
            if dists[v] is None or dists[u] + w < dists[v]:
                dists[v] = dists[u] + w
    return dists
```

🔷 Summary (for test)

- **DAGs**: directed graphs without cycles, generalize trees.
- **Toposort**: ordering where parents come before children.
  - Kahn's (indegree queue) or DFS (finish times).
- **Shortest Path in DAGs**:
  - Use topological order to relax edges in one pass.
  - Complexity = **O(V+E)**.
  - Works because *every prefix of a shortest path is itself shortest*, and topo order guarantees prerequisites are ready.

Would you like me to also add a **worked example DAG** (with edge weights, topo order, and step-by-step distance table) so you can quickly reference the shortest path process in your test?