Module 9 of 12

Topological sort and DAGs

Kahn's algorithm with in-degrees, a min-heap for the smallest order, the cycle test by count, and dynamic programming in topological order: longest paths, path counts, semesters.

Whenever a statement says "a must come before b" — courses and prerequisites, build steps, tasks with dependencies — the graph is directed and the question is about a topological order. One exists exactly when there is no cycle, and Kahn's algorithm finds one, or proves there is none, in O(V + E).

The lesson animates Kahn's algorithm peeling six courses with a min-heap — the in-degrees falling, each course joining the ready set the moment it has nothing left to wait for — and then turns the order into a schedule for dynamic programming: longest path, number of paths and fewest semesters are each one relaxation per edge. A game puts the algorithm's steps back in order.

The exercises print the lexicographically smallest course order or `IMPOSSIBLE`, and the longest weighted path in a DAG. The timed checkpoint asks for the fewest semesters and the number of paths from 1 to n modulo 10^9 + 7.

CONTENTS

1	Topological sort — Kahn's algorithm and DP on a DAG	14 min
✓	Checkpoint — Topological sort and DAGs	45 min

Exercises in this module compile on Clang 18 · C++20. Reference solutions and the answer key follow each lesson.

LESSON 1 · 14 MIN

Topological sort — Kahn's algorithm and DP on a DAG

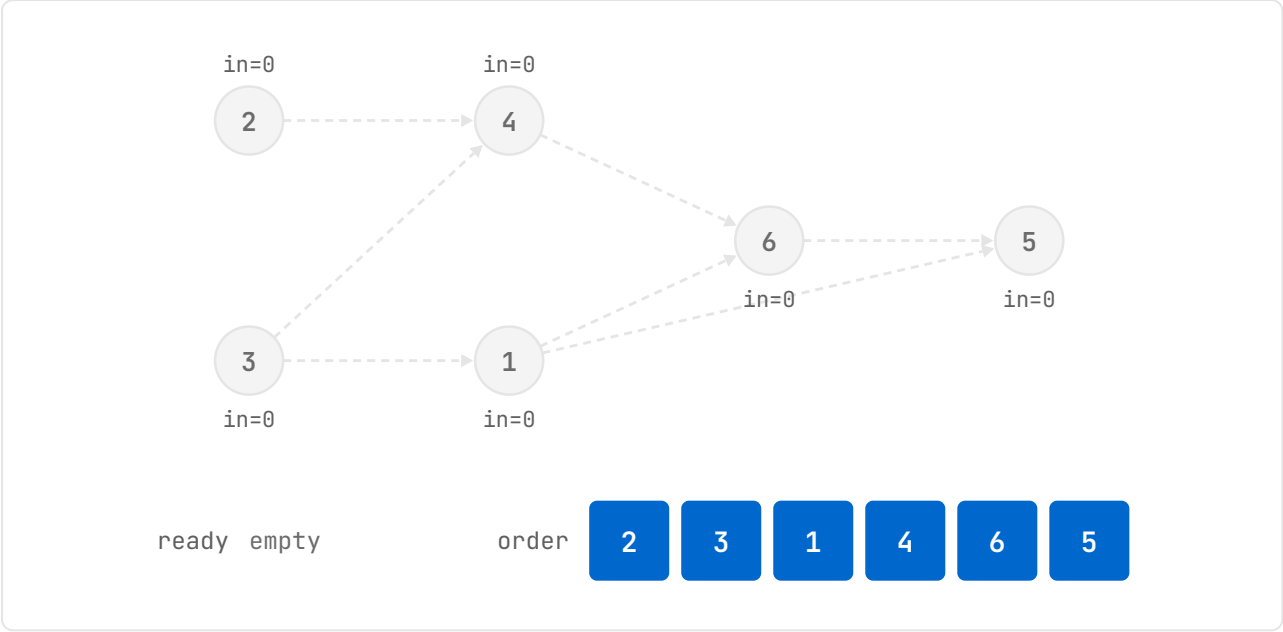
A **topological order** lists a DAG's vertices so every edge `a → b` has `a` before `b` — prerequisites before courses, tasks before what depends on them. It exists **exactly when there is no cycle**.

Kahn's algorithm

- Count `indeg[v]`, the number of edges into `v`.
- Every vertex with `indeg == 0` is **ready**: nothing has to come before it.
- Take a ready vertex, append it, and cross off its out-edges; a neighbour that drops to 0 becomes ready.
- Ready set a `queue` → any valid order. A **min-heap** → the lexicographically smallest one.

```
priority_queue<int, vector<int>, greater<>> ready;
for (int v = 1; v ⩽ n; ++v) if (indeg[v] == 0) ready.push(v);
vector<int> order;
while (!ready.empty()) {
    int u = ready.top(); ready.pop();
    order.push_back(u);
    for (int v : adj[u]) if (--indeg[v] == 0) ready.push(v);
}
bool cycle = (int)order.size() < n;
```

Kahn's algorithm peels the courses with nothing left to wait for



1. Count the arrows into each course: 1 has 1, 2 has 0, 3 has 0, 4 has 2, 5 has 2, 6 has 2. Courses 2 and 3 need nothing first, so they start in the ready min-heap.
2. Pop 2, the smallest ready course, and append it to the order. The heap hands out the smallest ready course, which makes the order the lexicographically smallest one. Cross off its arrows: 4 drops to 1.
3. Pop 3, the smallest ready course, and append it to the order. Cross off its arrows: 1 drops to 0, 4 drops to 0. 1 and 4 reach 0 and join the heap.
4. Pop 1, the smallest ready course, and append it to the order. Cross off its arrows: 6 drops to 1, 5 drops to 1.
5. Pop 4, the smallest ready course, and append it to the order. Cross off its arrows: 6 drops to 0. 6 reaches 0 and joins the heap.
6. Pop 6, the smallest ready course, and append it to the order. Cross off its arrows: 5 drops to 0. 5 reaches 0 and joins the heap.
7. Pop 5, the smallest ready course, and append it to the order. It has no arrows out.
8. The heap is empty and all 6 courses are placed: 2, 3, 1, 4, 6, 5. Every arrow points forwards in it. Had fewer than 6 been placed, the rest would sit on a cycle.

Cycles

- On a cycle every vertex waits for another of the cycle, so none reaches 0.
- `order.size() < n` **is** the cycle test — no separate DFS needed.

DP on a DAG

The order is a schedule for dynamic programming: when `u` comes up, every path into it is already counted.

- Longest path: `best[v] = max(best[v], best[u] + w)` over edges in order — O(V + E), though NP-hard on general graphs.
- Number of paths `1 → n`: `ways[v] += ways[u]`, mod 10^9 + 7.
- Fewest semesters (any number of courses at once): `level[v] = max(level[v], level[u] + 1)`.

GAME · PUT IN ORDER

Kahn's algorithm, step by step

Put the steps of Kahn's algorithm in order.

Tap one step, then another, to swap them — or use the arrows.

1. Repeat until the ready set is empty
2. Count the in-degree of every vertex
3. If fewer than V vertices were placed, report a cycle
4. Put every vertex with in-degree 0 into the ready set
5. Take a ready vertex and append it to the order
6. Lower each out-neighbour's in-degree; add those that reach 0

Check

Pitfalls

- A queue where the statement wants the **smallest** order — use the min-heap.
- Recursing in a DFS topo sort on a 10^5-vertex chain: stack overflow. Kahn's is iterative.
- Longest path with `int`: 10^5 edges of 10^9 needs `long long`.

More practice

- Course Schedule
- Course Schedule II

- Parallel Courses
- Find Eventual Safe States
- Largest Color Value in a Directed Graph

EXERCISE 1

The smallest course order

There are `n` courses `1` to `n` and `m` rules `a b`, each meaning course `a` must be taken before course `b`. Print an order that takes every course after all of its prerequisites; among all such orders, print the lexicographically smallest (compare the first course, then the second, …). Print `IMPOSSIBLE` if no order exists.

Input: `n m`, then `m` lines `a b` (`1 ≤ n ≤ 10^5`, `0 ≤ m ≤ 2·10^5`, `a ≠ b`). **Output:** the `n` courses on one line separated by spaces, or `IMPOSSIBLE`.

```
6 7
3 1
3 4
2 4
4 6
1 6
6 5
1 5
```

prints

```
2 3 1 4 6 5
```

INPUT 1

```
6 7
3 1
3 4
2 4
4 6
1 6
6 5
1 5
```

OUTPUT 1

```
2 3 1 4 6 5
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<vector<int>> adj(n + 1);
    vector<int> indeg(n + 1, 0);
    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        ++indeg[b];
    }
    priority_queue<int, vector<int>, greater<>> ready;
    for (int v = 1; v <= n; ++v)
        if (indeg[v] == 0) ready.push(v);
    vector<int> order;
    order.reserve(n);
    while (!ready.empty()) {
        int u = ready.top();
        ready.pop();
        order.push_back(u);
        for (int v : adj[u])
            if (--indeg[v] == 0) ready.push(v);
    }
    if ((int)order.size() < n) {
        cout << "IMPOSSIBLE\n";
        return 0;
    }
    for (int i = 0; i < n; ++i) cout << order[i] << (i + 1 == n ? '\n' : ' ');
    return 0;
}
```

EXERCISE 2

The longest path in a DAG

A directed acyclic graph has vertices `1` to `n` and `m` weighted edges. Print the largest total weight of any path (a path may start and end anywhere; a single vertex is a path of weight `0`).

Input: `n m`, then `m` lines `u v w` (`1 ≤ n ≤ 10^5`, `0 ≤ m ≤ 2·10^5`, `1 ≤ w ≤ 10^9`). The graph has no cycle. **Output:** one integer.

```
5 6
1 2 3
1 3 2
2 4 4
3 4 6
4 5 1
2 5 2
```

prints

```
9
```

The path `1 → 3 → 4 → 5` weighs 2 + 6 + 1.

INPUT 1

```
5 6
1 2 3
1 3 2
2 4 4
3 4 6
4 5 1
2 5 2
```

OUTPUT 1

```
9
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<vector<pair<int, ll>>> adj(n + 1);
    vector<int> indeg(n + 1, 0);
    for (int i = 0; i < m; ++i) {
        int u, v;
        ll w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        ++indeg[v];
    }
    queue<int> q;
    for (int v = 1; v <= n; ++v)
        if (indeg[v] == 0) q.push(v);
    vector<ll> best(n + 1, 0);
    ll answer = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        answer = max(answer, best[u]);
        for (auto [v, w] : adj[u]) {
            best[v] = max(best[v], best[u] + w);
            if (--indeg[v] == 0) q.push(v);
        }
    }
    cout << answer << '\n';
    return 0;
}
```

QUIZ

1. A directed graph has a topological order exactly when…

A. it has no cycle

B. it is connected

C. every vertex has an out-edge

D. it has no vertex of in-degree 0

2. Kahn's algorithm placed 7 of 10 vertices and the ready set is empty. What does that mean?

A. The graph has a cycle

B. Three vertices are isolated

C. The order is still valid

D. The heap was the wrong type

3. Which ready set makes Kahn's algorithm print the lexicographically smallest order?

A. A FIFO queue

B. A stack

C. A min-heap

D. A max-heap

4. Edges `1→2 (5)`, `1→3 (1)`, `3→2 (7)`. What is the longest path weight from 1?

A. 5

B. 7

C. 8

D. 13

ANSWER KEY · TOPOLOGICAL SORT — KAHN'S ALGORITHM AND DP ON A DAG

1. A On a cycle each vertex must come before the next, which is impossible all the way round; without a cycle Kahn's algorithm always finds an order. Connectivity is irrelevant, and a DAG always has an in-degree-0 vertex.

2. A Isolated vertices have in-degree 0 and would have been placed. The three left over each still wait for an edge from another leftover — they lie on a cycle or behind one.

3. C Taking the smallest ready vertex every time is the greedy choice that makes each position as small as possible. A queue or stack gives some valid order, not necessarily the smallest.

4. C `1 → 3 → 2` weighs 1 + 7 = 8, more than `1 → 2` at 5. Relaxing in the order 1, 3, 2 finds it because 3 comes before 2.

CHECKPOINT · 45 MIN

Checkpoint — Topological sort and DAGs

A 45-minute timed checkpoint on module 9: topological orders, Kahn's algorithm and its cycle test, and dynamic programming in topological order.

How it works. Ten questions and two programs, in 45 minutes from when you start the timer. 70% on the questions and both programs accepted clears the module.

Before you start, make sure you can answer these from memory:

- When a topological order exists, and how Kahn's algorithm proves it does not.
- What a min-heap instead of a queue changes about the order.
- Why every path into `u` is final when `u` comes up in topological order.
- How longest path, number of paths and fewest semesters each become one relaxation per edge.

The programs are the fewest semesters to finish every course (or -1 on a cycle), and the number of paths from 1 to n in a DAG modulo 10^9 + 7.

EXERCISE 1

Fewest semesters

There are `n` courses `1` to `n` and `m` rules `a b` (course `a` must be finished in an earlier semester than course `b`). Any number of courses can be taken in one semester. Print the fewest semesters needed to finish every course, or `-1` if it is impossible.

Input: `n m`, then `m` lines `a b` (`1 ≤ n ≤ 10^5`, `0 ≤ m ≤ 2·10^5`, `a ≠ b`). **Output:** one integer.

```
5 4
1 3
2 3
3 4
2 5
```

prints

```
3
```

Semester 1: courses 1 and 2; semester 2: 3 and 5; semester 3: 4.

INPUT 1

```
5 4
1 3
2 3
3 4
2 5
```

OUTPUT 1

```
3
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<vector<int>> adj(n + 1);
    vector<int> indeg(n + 1, 0);
    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        ++indeg[b];
    }
    queue<int> q;
    vector<int> level(n + 1, 1);
    for (int v = 1; v <= n; ++v)
        if (indeg[v] == 0) q.push(v);
    int placed = 0, answer = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        ++placed;
        answer = max(answer, level[u]);
        for (int v : adj[u]) {
            level[v] = max(level[v], level[u] + 1);
            if (--indeg[v] == 0) q.push(v);
        }
    }
    cout << (placed < n ? -1 : answer) << '\n';
    return 0;
}
```

EXERCISE 2

Count the paths

A directed acyclic graph has vertices `1` to `n` and `m` edges (no two edges are the same). Print the number of different paths from vertex `1` to vertex `n`, modulo `10^9 + 7`. (With `n = 1` there is one path, the empty one.)

Input: `n m`, then `m` lines `u v` (`1 ≤ n ≤ 10^5`, `0 ≤ m ≤ 2·10^5`). The graph has no cycle. **Output:** one integer.

```
5 6
1 2
1 3
2 4
3 4
4 5
2 5
```

prints

```
3
```

The paths are `1-2-4-5`, `1-3-4-5` and `1-2-5`.

INPUT 1

```
5 6
1 2
1 3
2 4
3 4
4 5
2 5
```

OUTPUT 1

```
3
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll MOD = 1'000'000'007;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<vector<int>> adj(n + 1);
    vector<int> indeg(n + 1, 0);
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        ++indeg[v];
    }
    queue<int> q;
    for (int v = 1; v <= n; ++v)
        if (indeg[v] == 0) q.push(v);
    vector<ll> ways(n + 1, 0);
    ways[1] = 1;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : adj[u]) {
            ways[v] = (ways[v] + ways[u]) % MOD;
            if (--indeg[v] == 0) q.push(v);
        }
    }
    cout << ways[n] << '\n';
    return 0;
}
```

CHECKPOINT QUESTIONS

1. What is the time complexity of Kahn's algorithm with a FIFO queue?

A. O(V + E)

B. O(V log V)

C. O(V · E)

D. O(E log E)

2. Why does a vertex on a cycle never reach in-degree 0 in Kahn's algorithm?

A. Its in-edge from the previous cycle vertex is only removed when that vertex is placed, and that vertex waits the same way

B. Kahn's algorithm skips cycles on purpose

C. Cycle vertices have no in-edges

D. The queue overflows

3. Rules `1→2`, `3→2`. Which order does Kahn's algorithm with a min-heap print?

A. 1 3 2

B. 3 1 2

C. 1 2 3

D. 2 1 3

4. Longest path is NP-hard on general graphs. Why is it O(V + E) on a DAG?

A. In topological order every path into u is complete before u's edges are relaxed

B. DAGs have at most V edges

C. All DAG edges have weight 1

D. Dijkstra works for longest paths on DAGs

5. Rules `1→2`, `2→3`, `1→3`. Fewest semesters to take all three?

A. 1

B. 2

C. 3

D. Impossible

6. A DFS-based topological sort on a chain of 10^5 vertices crashes. The most likely cause?

A. Stack overflow from recursion depth 10^5

B. Integer overflow

C. The graph has a cycle

D. The heap is too big

7. Counting paths `1 → n` in a DAG, which update is right for edge `u → v`?

A. ways[v] += ways[u]

B. ways[v] = max(ways[v], ways[u] + 1)

C. ways[v] = ways[u]

D. ways[u] += ways[v]

8. Kahn's algorithm, edges `5→1`, `5→2`, `1→2`, FIFO queue. What is `indeg[2]` at the start?

A. 0

B. 1

C. 2

D. 3

9. Which problem is NOT a topological-sort problem?

A. Order build steps with dependencies

B. Check whether course prerequisites are satisfiable

C. Find the shortest path in an undirected weighted graph with cycles

D. Count paths in a DAG

10. You need any valid order and n = 10^6. Queue or min-heap?

A. A queue: O(V + E), and any order is accepted

B. A min-heap: it is always required

C. Either costs the same

D. Neither works at that size

ANSWER KEY · CHECKPOINT — TOPOLOGICAL SORT AND DAGS

1. A Each vertex is pushed and popped once and each edge decrements one in-degree once. A min-heap adds a log V factor.

2. A Every vertex of the cycle waits for its predecessor on the cycle, which waits for its own — no one goes first, so the count of placed vertices ends short.

3. A 1 and 3 start ready; the heap gives 1 first, then 3, which brings 2's in-degree to 0. `1 2 3` breaks the rule 3 → 2.

4. A The order is a schedule for DP: best[u] is final when u comes up, so one relaxation per edge suffices. DAGs can have many edges, any weights, and Dijkstra's greedy settling does not work for maximisation.

5. C 3 needs 2, which needs 1: levels 1, 2, 3. The extra rule 1 → 3 changes nothing.

6. A Recursive DFS goes one frame per vertex down the chain; a default stack often cannot hold 10^5 frames. Kahn's algorithm is iterative and has no such limit.

7. A Each path to u extends by the edge into a path to v, and paths through different last edges are different, so the counts add.

8. C Two edges point into 2: from 5 and from 1.

9. C Undirected graphs with cycles have no topological order; that is Dijkstra's territory. The others are all orders or DP over a DAG.

10. A Only the lexicographically smallest order needs the heap and its log factor. When any order is accepted, a plain queue is faster.