Final Interview Report – Course Scheduling Problem

---

1. Problem Description

I presented the following problem to all interviewees:

> There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`.
> You are given an array `prerequisites` where `prerequisites[i] = [a, b]` indicates that you must take course `b` before course `a`.
> Return the ordering of courses you should take to finish all courses. If there are multiple valid answers, return any of them. If it is impossible to finish all courses, return an empty array.

Examples

Example 1
Input: `numCourses = 2`, `prerequisites = [[1, 0]]`
Output: `[0, 1]`
Explanation: To take course 1, you must first take course 0.

Example 2
Input: `numCourses = 4`, `prerequisites = [[1, 0], [2, 0], [3, 1], [3, 2]]`
Output: `[0, 1, 2, 3]` or `[0, 2, 1, 3]`
Explanation: Course 3 depends on courses 1 and 2, which both depend on course 0.

Example 3
Input: `numCourses = 1`, `prerequisites = []`
Output: `[0]`

Constraints

- $1 \leq$ `numCourses` ≤ 2000
- $0 \leq$ `prerequisites.length` $\leq$ `numCourses * (numCourses - 1)`
- All elements in `prerequisites[i]` are distinct and satisfy:
  - $0 \leq$ `a, b` $<$ `numCourses`
  - `a` $\neq$ `b`

---

2. Solution and Complexity

The problem is equivalent to finding a **topological ordering** in a **directed graph**. Each course is a node, and each prerequisite pair is a directed edge.

2.1 BFS – Kahn's Algorithm

1. Build an adjacency list and an in-degree array.

2. Initialize a queue with nodes that have in-degree 0.
3. Pop from the queue, reduce the in-degree of its neighbors.
4. If in-degree of a neighbor becomes 0, push it into the queue.
5. If all nodes are processed, return the ordering; else return an empty array.

Time Complexity: `O(V + E)`
Space Complexity: `O(V + E)`

2.2 DFS-Based Topological Sort

1. Build an adjacency list.
2. Use a visited status for each node:
    - 0 = unvisited, 1 = visiting, 2 = visited.
3. For each unvisited node, call DFS:
    - If you re-encounter a node marked "visiting", there's a cycle.
4. Append each node to the result list after exploring all its children.
5. Reverse the result list.

Time Complexity: `O(V + E)`
Space Complexity: `O(V + E)` including the recursion stack

---

3. Hints Evaluation

Interviewees were offered these hints progressively based on need:

1. **Model the problem as a graph**: Treat each course as a node and each prerequisite pair `[a, b]` as a directed edge from `b` to `a`. This means you need to take course `b` before `a`.

2. **Build the graph and compute in-degrees**: Create an adjacency list to represent the graph and an array to store the in-degree (number of incoming edges) for each course.

3. **Use Kahn's algorithm for topological sorting**: Initialize a queue with all nodes that have in-degree 0 (no prerequisites). Repeatedly remove a node from the queue, add it to the result, and reduce the in-degree of its neighbors. If any neighbor's in-degree becomes 0, add it to the queue.

4. **Detect cycles**: If the number of courses in the result is less than `numCourses`, there must be a cycle in the graph, making it impossible to finish all courses.

---

4. Skills and Knowledge Assessed

This problem was effective for evaluating the following areas:

- Depth-First Search
- Breadth-First Search
- Graph
- Topological Sort

---

5. Common Mistakes

- Difficulty recognizing the problem as a classic graph modeling task.
- Lack of understanding or recall of the **in-degree** concept and how it applies to topological sorting.
- Incorrect edge direction: many students created edges from the dependent course to the prerequisite, reversing the correct logic (`b → a`).
- Confusion between the graph structure and the in-degree tracking — few realized these could and should be maintained in separate data structures.

---

6. General Observations

In most interviews, students only began to make progress after realizing the problem could be modeled as a graph. This realization was not immediate for many, who initially attempted linear or ad-hoc approaches, often getting stuck on problem interpretation.

Even after identifying the graph structure, several students struggled to correctly implement BFS or DFS. A common issue was initializing the queue or stack with **all** nodes that had `in-degree == 0`; many added only one.

Performance varied significantly depending on prior familiarity with graph algorithms. Those with a strong grasp advanced quickly once the model was clear.

---

7. Evaluation of Problem Reuse Across Interviews

Reusing this problem proved effective in differentiating levels of graph algorithm proficiency. Students with solid understanding solved it with little assistance, while others required multiple hints, even after identifying it as a graph problem.

---

8. Noteworthy Interview Moments

One student solved the problem without using standard auxiliary structures like a queue or stack. Instead, they repeatedly traversed the course list, identifying and processing nodes with `in-degree == 0`.

No student tried the DFS-Based Topological Sort solution, they all went for BFS – Kahn’s Algorithm.