Overview of Kahn's Algorithm

Report Issue

When selecting courses for the next semester in college, you might have noticed that some advanced courses have prerequisites that require you to take some introductory courses first. In Figure 12, for example, to take Course C, you need to complete Course B first, and to take Course B, you need to complete Course A first. There are many courses that you must complete for an academic degree. You do not want to find out in the last semester that you have not completed some prerequisite courses for an advanced course. So, how can we arrange the order of the courses adequately while considering these prerequisite relationships between them?

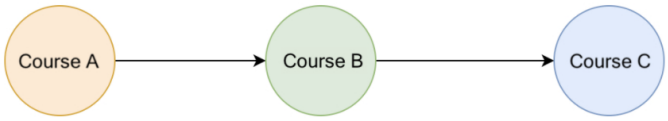
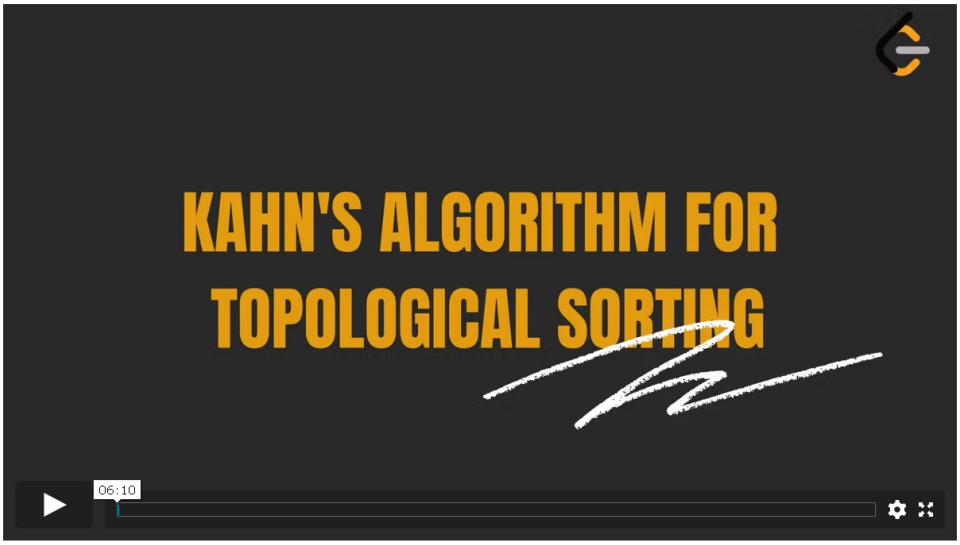


Figure 12. Prerequisite relationships between courses

"Topological sorting" helps solve the problem. It provides a linear sorting based on the required ordering between vertices in directed acyclic graphs. To be specific, given vertices `u` and `v`, to reach vertex `v`, we must have reached vertex `u` first. In "topological sorting", `u` has to appear before `v` in the ordering. The most popular algorithm for "topological sorting" is Kahn's algorithm.

Video Explanation

In this video, we'll cover how the Kahn's Algorithm can be used for topological sorting.



Note, for simplicity while introducing Kahn's algorithm, we iterated over all of the courses and reduced the in-degree of those for which the current course is a prerequisite. This requires us to iterate over all E prerequisites for all V courses resulting in $O(V \cdot E)$ time complexity at the cost of $O(V)$ space to store the in degree for each vertex.

However, this step can be performed more efficiently by creating an adjacency list where `adjacencyList[course]` contains a list of courses that depend on `course`. Then when each course is taken, we will only iterate over the courses that have the current course as a prerequisite. This will reduce the total time complexity to $O(V + E)$ at the cost of an additional $O(E)$ space to store the adjacency list.

Limitation of the Algorithm

- "Topological sorting" only works with graphs that are directed and acyclic.
- There must be at least one vertex in the "graph" with an "in-degree" of 0. If all vertices in the "graph" have a non-zero "in-degree", then all vertices need at least one vertex as a predecessor. In this case, no vertex can serve as the starting vertex.

Complexity Analysis

V represents the number of vertices, and E represents the number of edges.

- Time Complexity: $O(V + E)$.
    - First, we will build an adjacency list. This allows us to efficiently check which courses depend on each prerequisite course. Building the adjacency list will take $O(E)$ time, as we must iterate over all edges.
    - Next, we will repeatedly visit each course (vertex) with an in-degree of zero and decrement the in-degree of all courses that have this course as a prerequisite (outgoing edges). In the worst-case scenario, we will visit every vertex and decrement every outgoing edge once. Thus, this part will take $O(V + E)$ time.
    - Therefore, the total time complexity is $O(E) + O(V + E) = O(V + E)$.
- Space Complexity: $O(V + E)$.
    - The adjacency list uses $O(E)$ space.
    - Storing the in-degree for each vertex requires $O(V)$ space.
    - The queue can contain at most V nodes, so the queue also requires $O(V)$ space.