Task1(A)

I use DFS to solve this problem and iterates through each course, checking if it forms a cycle or not. If a cycle is found, it means there are no possible sequences, so it return Impossible. If no cycle is found, I apply topological sort by DFS and reverse the resulting stack to ~~get th~~ ensure that each course is taken after all of it's prerequisites.

Task1(B)

I use Kahn's Algorithm to * solve this problem. This algorithm use the concept of in-degree. While constructing adjacency list, I also keep track in-degree of every course. Firstly, I enqueue courses with an in-degree of 0 because in-degree 0 means it has no prerequisite. Then, iteratively process the queue and reduce the in-degree of adjacent courses. and enqueue those with an in-degree of 0. It continues untill all courses are processed or a cycle is detected. If a cycle is detected, it means ~~no~~ there are no valid sequences. Otherwise, it will provide a feasible sequence.

Task2

~~In this prob~~

To solve this problem, I followed the same approach of Task1B. Just I use a heapq instead of a regular queue to prioritize courses with lower numbers. It ensures that I explore courses in lexicographically order.

Task3

I use kosaraju's Algorithm to solve this problem which is based on DFS, Firstly, I run DFS on the original graph and get a topological order, Then, I reverse the graph and pop element from the stack. Now, run DFS on poped element but here I run DFS on the Reversed graph and get the strongly connected components.