1.a. This code solves the problem of finding a topological order using DFS. It maintains an array named "topological_order" to store the sequence of nodes in the topological order. Once a node is fully explored, it is added to the end of the topological order array after all its adjacent nodes have been explored. Additionally, the code includes a cycle detection to ensure that the graph is acyclic. Then prints the topological order in an output file.

1.b. This code solves the problem of finding a topological order using BFS. First, all the nodes with an indegree of 0 are placed in a queue. Then, one by one, nodes are dequeued and their adjacent nodes' indegrees are decreased. If any adjacent node's indegree becomes 0, it is also added to the queue. This process continues until the queue becomes empty.

2. This code works the same as task1b. Here, we are just using a built-in priority queue to sort it into a lexicographically smallest valid course sequence.

3. This code solves the strongly connected component problem. First, it maintains a stack. Once a node is fully visited, it is appended to the stack. Then, the entire graph is reversed. After that, we iterate through the stack one by one, adding nodes to a strongly connected component as long as they can be reached in the reversed graph. We use a checker to determine if a node is already within another node's strongly connected list or not.