Task 1:
DFS approach explanation:
To solve this problem in DFS method I used modified version of DFS function here to detect if there is any cycle or not. For that I added path set to keep track of the nodes in the current DFS path. Is the neighbor already in the present path then there is a cycle and then DFS function will return True and finally the output will be IMPOSSIBLE. If there is no cycle then DFS function will work as normal and lastly in the output file correct DFS order will be shown.

BFS approach explanation:
To solve this problem in BFS method I made a function called bfs_order to get the correct BFS order. To do that I took two dictionary called graph and in_degree. In graph dictionary I appended prerequisites of every courses and in in_order dictionary I incremented by 1 if there is any prerequisites of any courses. Then I took a double ended queue called deque to perform BFS. Then BFS parts works as it is. And lastly if the returns the BFS order if all courses are included (i.e. no cycles) otherwise returns an empty list. Based on that the output file show the result.

Task 2:
To solve the task1 problem lexicographically there should be a little difference in the BFS part of the BFS method of task 1. Instead of popleft I used min to get the minimum data from the queue. And rest of the parts are same as task 1 BFS method.

Task 3:
It starts by reading the input file, which contains the number of vertices and edges, and builds the graph representation based on prerequisite requirements. The code then performs a topological sort on the graph using depth-first search (DFS) to obtain the order of vertices in a stack. After that, it transposes the graph and runs DFS on the transposed graph to find the SCCs. It uses a stack-based approach to keep track of visited vertices and their order. Then it shows the corresponding output.