**Task 1(a):** This code reads a directed graph's information from an input file ('input1a.txt'). It checks for the existence of cycles in the graph using a cycle-detection algorithm. If a cycle is detected, it writes 'IMPOSSIBLE' to an output file ('output1a.txt'). Otherwise, it performs a depth-first search traversal to obtain a topological ordering of the graph's nodes and writes the result to the output file. The code efficiently handles cycle detection and topological sorting in directed graphs.

**Task 1(b):** This code reads a directed graph's information from an input file ('input1b.txt') and performs a topological sorting using Breadth-First Search (BFS). It constructs the graph and calculates the indegree of each node. The BFS-based topological sort is performed by iteratively selecting nodes with zero indegree, reducing indegrees of their neighbors, and appending them to the result. The code then checks if the result size matches the number of nodes in the graph to verify successful sorting. If successful, it writes the sorted order to an output file ('output1b.txt'); otherwise, it writes 'IMPOSSIBLE' indicating a cycle or sorting failure. The code efficiently handles topological sorting and cycle detection in directed graphs.

**Task 2:** This code reads information about a directed graph from an input file ('input2.txt'). It constructs the graph and calculates the indegree of each node. The code then checks for the existence of cycles using a DFS-based cycle detection function. If a cycle is found, it writes 'IMPOSSIBLE' to an output file ('output2.txt'). Otherwise, it performs a BFS-based topological sort by selecting nodes with zero indegree, reducing indegrees of their neighbors, and appending them to the result. The result is then written to the output file. The code handles both cycle detection and topological sorting in a directed graph efficiently.

**Task 3:** This code reads information about a directed graph from an input file ('input3.txt'). It constructs two adjacency lists for the graph and its inverse. The code performs two Depth-First Search (DFS) traversals: one to obtain a topological ordering of the graph's nodes and another to find strongly connected components using the inverse adjacency list. It then prints each strongly connected component in a separate line to an output file ('output3.txt'). The code efficiently identifies strongly connected components and produces the correct output format.