Task 1(a)

First of all, lets create a dictionary where keys are the vertices means pre-requisite and values are the courses, means a list of courses which pre-req are same.

Then a track list to keep track of the number of prerequisites for each course.

Now perform DFS to the graph from each course with no prerequisites and not visited yet. For every vertex we will look for it's adj. list and decrement one from track list for every neighbour and last track the course's by adding result list.

Now, if the len of result is less than number of courses that mean not possible to do the courses. Otherwise print the list of taking courses.

Task 1 (d)

Same as before, but here we will use BFS and use queue rather of stack. The idea is same as of a.

Task 2

In previous problem we use stack and queue but in here we will use priority queue because here we need to determined the order of tasks to perform an their dependencies.

First, we import heap and initial a priority queue list. Then, we use heapify which converts the priority queue list to a heap. Then, we make bfs and by the same way from task one we do the

remain nothing. But there it will be ensured that at time the lowest value will be prioritize so that if we poped the tasks with smallest dependency from the queue. And then added to the result. And, same as before If the len of result is less then number of courses then it will not be possible other wise we print the result result list.

Task 3:

For sec, we and will do;

(i) ~~BFS~~ DFS

(ii) reverse the edges

(iii) Again DFS

In this task, we will create two dictionary one is vertices connecting to the another and other is the reverse.

Now we will do the DFS for every vertices Now, in stack list we will add the vertices if top-sort parameter as True which is True at first. It append to a stack reverse order of finishing time. and parameter as False if finds the strogly connected components by traversing the reverse graph and appending the veitices to the strugL.

Now, while stack is not empty, we pop vertices from the stack and again run DFS on the reverse graph to the find the SCC's. Now, from strong list we will traverse each inner group and print it.