Title

Reasoning

We can first make a graph of the courses and their prerequisites by making an adjacency list where $adj[u]$ is the list of courses that u is a prereq for. We can also track the in-degree for each course as a variable. We can then do a topological sort, giving us the classes in an order such that every class will be proceeded by it prerequisites. After this, we can go through and calculate the maximum number of sequential classes using dynamic programming.

Psudocode

```
def longest_prereq_chain(n, prerequisites):
    graph = adjacency list of size n
    indegree = array of size n
    for deg in indegree: deg = 0 # O(n)
    # graph
    for course i in [1..n]: # O(n+m) (goes through each courses
prerequisites for each course)
        for prereq in prerequisites[i]:
            graph[prereq].append(i)
            indegree[i] += 1

    # Topolgical sort
    queue = all courses with indegree == 0 # O (n)
    S = array of size n, initialized to 1 # O(n)

    while queue not empty: # Each course is done once, runs n times
        u = queue.pop()
        for v in graph[u]: # Runs deg(n) times or O(m ) total
            S[v] = max(S[v], S[u] + 1)
            indegree[v] -= 1
            if indegree[v] == 0:
                queue.push(v)

    return max(S)# O(n)
```

Proof

This algorithm works because in the topologically sorted array all classes will appear after their prereqs. This means that when we do the dynamic programming section, we can start at the beginning and move through, adding the maximum length of the chain to $S[j]$. Assuming that all previous values of S are correct, the current value will be the maximum of it's prerequisites plus one. If a class has no prerequisites, its value in S is 1. For the psudocode, I simplified it by moving the dynamic programming into the loop for topological sort, since we can handle each element as soon as it is sorted, reducing complexity.

Running Time

Estimate

The running time is $O(n+m)$

Running Time Reasoning

As seen from the comments, the running time is $O(n)+O(n+m)+O(m+n)+O(n)$ or $O(n+m)$