دنباله‌ای معتبر از دوره‌ها با توجه به پیش‌نیازی

برای پیدا کردن دنباله‌ای معتبر از دوره‌ها به طوری که قبل از دیدن هر دوره، تمام پیش‌نیاز هایش دیده شده باشد، ما می‌توانیم مسأله را با یک گراف جهت‌دار غیر مدور(DAG) مدل کنیم و سپس با استفاده از Topological Sorting به یک جواب با شرایط مسأله برسیم.

هر گره در گراف نشان‌دهده یک دوره است و یک یال جهت‌دار از گره A به گره B می‌کشیم اگر و تنها اگر A پیش نیاز B باشد.

مراحل حل مسأله به این صورت است که ابتدا با توجه به نکات گفته شده گراف را می‌سازیم و همچنین تعداد یال‌های ورودی هر راس که نشان‌دهنده تعداد پیش نیاز های دوره هست را نیز نگه می‌داریم. سپس گره‌هایی که هیچ پیش نیازی ندارد یعنی تعداد یال‌های وارد شده به آن‌ها صفر است را مشخص می‌کنیم و به یک صف اضافه میکنیم.(لازم به ذکر است که می‌توانستم از رویکرد DFS استفاده کرده و اینجا به جای صف یک پشته تعریف می‌کردیم) حالا تا زمانی که صف خالی نشده است، هر بار از صف یک گره را dequeue کرده و می‌توانیم آن را خروجی دهیم و از هر کدام از همسایه های آن گره تعداد پیش نیاز ها(تعداد یال‌های ورودی) را یک واحد کم کرده و چک می‌کنیم اگر پیش نیاز های گره‌ای برابر صفر شد، آن را به صف enqueue می‌کنیم.

بعد از خالی شدن صف چک می‌کنیم که اگر هنوز گره‌ای با تعداد یال‌های ورودی مخالف صفر وجود داشت یعنی در گرافمان دور وجود داشته و هیچ دنباله‌ی معتبری وجود ندارد.

```
Input: n as number of courses and m as number of prerequisites. Then in
each line we have “a b” meaning a is prerequisite of b.
Output: a valid sequence of courses
```

```
n, m = get input

graph = empty adjacency list
in_degree = an array of zeros with size of n

# Build graph and in_degree array
for i=0 to m-1:
      a, b = get input

      if a does not exist in graph:
            add node a to graph
      if b does not exist in graph:
            add node b to graph

      add b to adjacency list of node a # add a → b
      increment in_degree of b by 1

#find courses with no prerequisite(zero in_degree)
queue = empty queue
for each node in graph:
      if in_degree of node == 0:
            add node to the queue
```

```
#perform topological sort
sequence = empty list
while queue is not empty:
      node = queue.dequeue()
      add node to sequence
      for each neighbor of node in graph:
            decrement in_degree of the neighbor by 1
            if in_degree of the neighbor == 0:
                  add the neighbor to the queue

#check cycles
if sequence.size() == n:
      return sequence
else
      return empty list #there is cycles. no valid sequence.
```