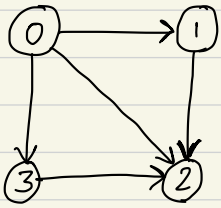
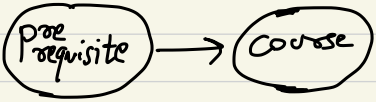
COURSE SCHEDULE



CLASSIC APPLICATION OF TOPOLOGICAL SORT.
→ RESOLUTION OF DEPENDENCIES

Input:
[[1,0],[2,0],[3,0]
[2,1],[2,3]]

format:
[course, prerequisite]



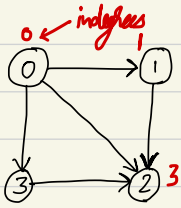
CREATE DEPENDENCY GRAPH.

GRAPH
0: [1, 2, 3]
1: [2]
2: []
3: [2]

tracks outgoing edges.

WE ALSO NEED TO TRACK, SO TO KNOW ALL PREREQUISITES HAVE BEEN MET.

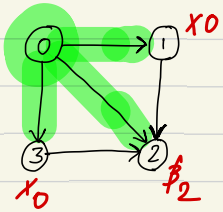
HOW!!
INDEGREE COUNT.
0: 0 → Node 0, No Prerequisites
1: 1
2: 3 → Node 2, 3 Prerequisites
3: 1
q{0, 1, 3}



We can only process nodes that have no dependencies.
i.e. indegree = 0.

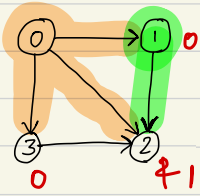
q [0]

Queue would only hold tasks that can be processed right away ⇒ indegree = 0

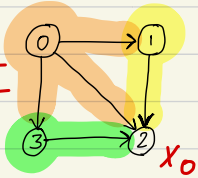


As we can see, as Node 0 is processed, outgoing dependencies are resolved.

q [1, 3]



q [3]



q [2]

IF WE ARE ABLE TO PROCESS ALL NODES
⇒ TOPOLOGICAL ORDERING EXISTS.

ALGORITHM:

```
G ← createGraph(); indegree ← computeIndegree();
q ← addIndegreeZeroTasks(); count ← N
while (!q.isEmpty())
    prereq ← q.poll(); count--;
    courses ← graph.get(prereq)
    for c : courses
        indegree[c]--
        if (indegree[c] == 0) q.add(c)
return count == 0;
```

YASH PRADHAN
STUDY NOTES