#LeetCode 210

https://leetcode.com/problems/course-schedule-ii/description/

Yifeng Zeng

#Description

210. Course Schedule II

#Idea Report

The first step is to find all the courses that has no prerequisites, these courses are the starting point. After finishing all the staring point, there must be some courses which prerequisites are these starting point, so then we can take those classes because we hava already finished their prerequisites. And starting from there we can take next step. So we can draw something like course X point to course Y then point to course Z. This is actually a directed graph each course is a node, and a directed graph has a very important propetry which is indegree, meaning how mainly nodes goes into the current node. So our starting point are the nodes with no indegrees which are the courses with no prerequisites. We then traverse the nodes with indegree == 0, and put them into a result array. Also when we traversed a node, we need to decrese the indegree of its neighbor by one, because we have already take the class meaning we finised one of the prerequisites of the neighbor class. So when the neighbor class's indegree becomes one, we know that that class can now be taken.

Code

```
public class Solution {

    public int[] findOrder(int n, int[][] pre) {
        List<Integer>[] neis = new List[n];
        for (int i = 0; i < n; i++) {
            neis[i] = new ArrayList<>();
        }

        int[] indegree = new int[n];
        for (int[] p : pre) {
            indegree[p[0]]++;
            neis[p[1]].add(p[0]);
```

```
        }

        Deque<Integer> q = new LinkedList<>();
        for (int i = 0; i < n; i++) {
            if (indegree[i] == 0) {
                q.offer(i);
            }
        }

        int[] res = new int[n];
        int index = 0;
        while (!q.isEmpty()) {
            int cur = q.poll();
            res[index++] = cur;
            for (int nei : neis[cur]) {
                indegree[nei]--;
                if (indegree[nei] == 0) {
                    q.offer(nei);
                }
            }
        }

        return index == n ? res : new int[0];
    }
}
```

#Summary

- Node-to-node reppresentation can be convertd to a directed graph.
- Topological sorting.
- List[] neis = new List[n]; is the new thing that I learned.