LeetCode 210

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

Yifeng Zeng

Description

210. Course Schedule II

There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

For example:

2, [[1,0]]

There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1]

4, [[1,0],[2,0],[3,1],[3,2]]

There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is [0,1,2,3]. Another correct ordering is[0,2,1,3].

Idea Report

Given some courses and some other other courses may be prerequisites. This can be represented as a directional graph. Each course is a node, and the prerequisite course points to another course means the edge from the prerequisite course to the other course. And we want to find out the order of taking these courses, which is basically a topological sorting of these courses. We can do both BFS (prefered) and DFS.

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 many nodes goes into the current node. So our starting point are the nodes with indegree equals to zeros 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 zero, we know that that class can now be taken.

Code

```
public class Solution {
    // BFS AC
    public int[] findOrder(int n, int[][] pre) {
        List<Integer>[] map = new List[n];
        int[] indegree = new int[n];
        for (int i = 0; i < n; i++) {
            map[i] = new ArrayList<>();
        }
        for (int[] p : pre) {
            map[p[1]].add(p[0]);
            indegree[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 next : map[cur]) {
                    indegree[next]--;
                    if (indegree[next] == 0) {
                        q.offer(next);
                    }
                }
            }

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

Summary

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

Follow Up

The DFS can also achieve topological sorting, but not really very intuitive. Basically for each node (course), we do a DFS and see if there is circle in the graph. If there is, then those courses cannot be finished, we return an empty array, otherwise we can return the order. The deepest node is the last node we want to take, so we can fill the output order array from the last index.

Code

```
public class Solution {
    // DFS with pruning, AC
    public int[] findOrder(int n, int[][] pre) {
        List<Integer>[] map = new List[n];
        for (int i = 0; i < n; i++) {
            map[i] = new ArrayList<>();
        }
        for (int[] p : pre) {
            map[p[1]].add(p[0]);
        }
```

```
        int[] visited = new int[n];
        int[] res = new int[n];
        int[] index = new int[1];
        index[0] = n - 1;
        for (int i = 0; i < n; i++) {
            if (visited[i] == -1) {
                continue;
            }
            if (findCircle(map, i, visited, res, index)) {
                return new int[0];
            }
        }

        return index[0] == -1 ? res : new int[0];
    }

    private boolean findCircle(List<Integer>[] map, int cur, int[] visited,
                               int[] res, int[] index) {
        if (visited[cur] == 1) {
            return true;
        }
        if (visited[cur] == -1) {
            return false;
        }

        visited[cur] = 1;
        for (int next : map[cur]) {
            if (visited[next] == -1) {
                continue;
            }
            if (findCircle(map, next, visited, res, index)) {
                return true;
            }
        }
        visited[cur] = -1;
        res[index[0]--] = cur;
        return false;
    }
}
```