LeetCode 210

Description

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].

Note:

The input prerequisites is a graph represented by a list of edges, not adjacency matrices.
Read more about how a graph is represented.
You may assume that there are no duplicate edges in the input prerequisites.

click to show more hints.

Hints:

This problem is equivalent to finding the topological order in a directed graph. If a cycle exists, no topological ordering exists and therefore it will be impossible to take all courses.
Topological Sort via DFS - A great video tutorial (21 minutes) on Coursera explaining the basic concepts of Topological Sort.
Topological sort could also be done via BFS.

Solution

```
public int[] findOrder(int numCourses, int[][] prerequisites) {
    if (numCourses == 0) return null;
    // Convert graph presentation from edges to indegree of adjacent list.
    int indegree[] = new int[numCourses], order[] = new int[numCourses], index = 0;
    for (int i = 0; i < prerequisites.length; i++) // Indegree - how many prerequis
        indegree[prerequisites[i][0]]++;

    Queue<Integer> queue = new LinkedList<Integer>();
    for (int i = 0; i < numCourses; i++)
        if (indegree[i] == 0) {
            // Add the course to the order because it has no prerequisites.
            order[index++] = i;
            queue.offer(i);
        }

    // How many courses don't need prerequisites.
    while (!queue.isEmpty()) {
        int prerequisite = queue.poll(); // Already finished this prerequisite cour
        for (int i = 0; i < prerequisites.length; i++)  {
            if (prerequisites[i][1] == prerequisite) {
                indegree[prerequisites[i][0]]--;
                if (indegree[prerequisites[i][0]] == 0) {
                    // If indegree is zero, then add the course to the order.
                    order[index++] = prerequisites[i][0];
                    queue.offer(prerequisites[i][0]);
                }
            }
        }
    }

    return (index == numCourses) ? order : new int[0];
}
```