Leetcode 210. Course Schedule II

Problem

Difficulty: Medium
There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [ai, bi]` indicates that you **must** take course `bi` first if you want to take course `ai`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return *the ordering of courses you should take to finish all courses*. If there are many valid answers, return **any** of them. If it is impossible to finish all courses, return **an empty array**.

Example 1:
Input: numCourses = 2, prerequisites = [1,0]
Output: [0,1]
Explanation: 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].
Example 2:
Input: numCourses = 4, prerequisites = [1,0],[2,0],[3,1],[3,2]
Output: [0,2,1,3]
Explanation: 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].
Example 3:
Input: numCourses = 1, prerequisites = []
Output: [0]

Constraints:

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= numCourses * (numCourses - 1)`
- `prerequisites[i].length == 2`
- `0 <= ai, bi < numCourses`
- `ai != bi`

- All the pairs `[ai, bi]` are **distinct**.

Solution

Understanding the problem:

- We're given a graph of n courses.
- Some courses have prerequisites.
- We want the ordering of courses that you should take to finish all courses.
- This order **MUST** account for prerequisites, and if such an order cannot be found then we return an empty array

Algorithm Description

This is essentially a topological sort that's from two arguments: numCourses, the number of course and prerequisites, an input array of prerequisites.

The format of this function is very similar to the standard pseudocode for topological sort with a few changes. It goes as follows:

1. Build an **adjacency list** from the given arguments
2. Create your arrays, (1) visited (2) cycled (3) output
    1. **visited** - boolean array that keeps track of whether or not a course has been visited
    2. **cycled** - boolean array that keeps track of whether or not a course is in a cycle
3. Declare an **inner function for DFS** similar to the dfs function in our topological sort pseudocode
    1. What's different though is that we only **pass the course** to it.
    2. Return *false if there's a **cycle**.
    3. Return *true* if a course has been **visited**.
    4. For every prerequisite of a course:
        1. Recurse and if that recursive call is false we have a cycle so we return false.
        2. Otherwise we **unmark** the course as **cycled**. Then, **mark** the course as visited. **Append** the current course to the output array. Before returning true.

Pseudocode

```
function plancourses(numCourse, prerequisites):
        build an adjacency list prereq from input array
        output = [0,...,0]
        visited = [false,...,false]
```

```
cycled = [false,...,false]
//inner dfs function below:
function dfs(course):
        if cycled[course] is true do:
                return false
        if visited[course] is true do:
                return true
        cycled[course] = true //marked as cycled
        for each class ∈ prereq[course] do:
                if dfs(course) is false:
                        return false
        cycled[course] = false //unmarked as cycled
        visited[course] = true //marked as visited
        output.append(course)
        return true
return order
```