0210. 课程表 II

ITCharge 大约 3 分钟

- 标签：深度优先搜索、广度优先搜索、图、拓扑排序
- 难度：中等

题目链接

- 0210. 课程表 II - 力扣

题目大意

描述：给定一个整数 $numCourses$，代表这学期必须选修的课程数量，课程编号为 $0 \sim numCourses - 1$。再给定一个数组 $prerequisites$ 表示先修课程关系，其中 $prerequisites[i] = [ai, bi]$ 表示如果要学习课程 ai 则必须要先完成课程 bi。

要求：返回学完所有课程所安排的学习顺序。如果有多个正确的顺序，只要返回其中一种即可。如果无法完成所有课程，则返回 组。

说明：

- $1 \leq numCourses \leq 2000$。
- $0 \leq prerequisites.length \leq numCourses \times (numCourses - 1)$。
- $prerequisites[i].length == 2$。
- $0 \leq ai, bi < numCourses$。
- $ai \neq bi$。
- 所有$[ai, bi]$ 互不相同。

示例：

- 示例 1：

```py
输入：numCourses = 2, prerequisites = [[1,0]]
输出：[0,1]
解释：总共有 2 门课程。要学习课程 1，你需要先完成课程 0。因此，正确的课程顺序为 [0,1]。
```

- 示例 2：

```py
输入：numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]
输出：[0,2,1,3]
解释：总共有 4 门课程。要学习课程 3，你应该先完成课程 1 和课程 2。并且课程 1 和课程 2 都应该排在课程 0 之后。
因此，一个正确的课程顺序是 [0,1,2,3] 。另一个正确的排序是 [0,2,1,3]。
```

解题思路

思路 1：拓扑排序

这道题是「0207. 课程表 」的升级版，只需要在上一题的基础上增加一个答案数组 $order$ 即可。

1. 使用哈希表 $graph$ 存放课程关系图，并统计每门课程节点的入度，存入入度列表 $indegrees$。
2. 借助队列 S，将所有入度为 0 的节点入队。
3. 从队列中选择一个节点 u，并将其加入到答案数组 $order$ 中。
4. 从图中删除该顶点 u，并且删除从 点出发的有向边 $< u, v >$（也就是把该顶点可达的顶点入度都减 1）。如果删除该边后顶点 v 的入度变为 0，则将其加入队列 S 中。
5. 重复上述步骤 $3 \sim 4$，直到队列中没有节点。
6. 最后判断总的顶点数和拓扑序列中的顶点数是否相等，如果相等，则返回答案数组 $order$ ，否则，返回空数组。

思路 1：代码

```py
import collections

class Solution:
    # 拓扑排序，graph 中包含所有顶点的有向边关系（包括无边顶点）
    def topologicalSortingKahn(self, graph: dict):
        indegrees = {u: 0 for u in graph}   # indegrees 用于记录所有顶点入度
        for u in graph:
            for v in graph[u]:
                indegrees[v] += 1           # 统计所有顶点入度

        # 将入度为 0 的顶点存入集合 S 中
```

```
        S = collections.deque([u for u in indegrees if indegrees[u] == 0])
        order = []                          # order 用于存储拓扑序列

        while S:
            u = S.pop()                     # 从集合中选择一个没有前驱的顶点 0
            order.append(u)                 # 将其输出到拓扑序列 order 中
            for v in graph[u]:              # 遍历顶点 u 的邻接顶点 v
                indegrees[v] -= 1           # 删除从顶点 u 出发的有向边
                if indegrees[v] == 0:       # 如果删除该边后顶点 v 的入度变为 0
                    S.append(v)             # 将其放入集合 S 中

        if len(indegrees) != len(order):    # 还有顶点未遍历（存在环），无法构成拓扑序列
            return []
        return order                        # 返回拓扑序列


    def findOrder(self, numCourses: int, prerequisites):
        graph = dict()
        for i in range(numCourses):
            graph[i] = []

        for v, u in prerequisite:
            graph[u].append(v)

        return self.topologicalSortingKahn(graph)
```

思路 1：复杂度分析

- **时间复杂度**：$O(n+m)$，其中 n 为课程数，m 为先修课程的要求数。
- **空间复杂度**：$O(n+m)$。