0802. 找到最终的安全状态

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

- 0802. 找到最终的安全状态 - 力扣

题目大意

描述：给定一个有向图 $graph$，其中 $graph[i]$ 是与节点 i 相邻的节点列表，意味着从节点 i 到节点 $graph[i]$ 中的每个节点都有一条有向边。

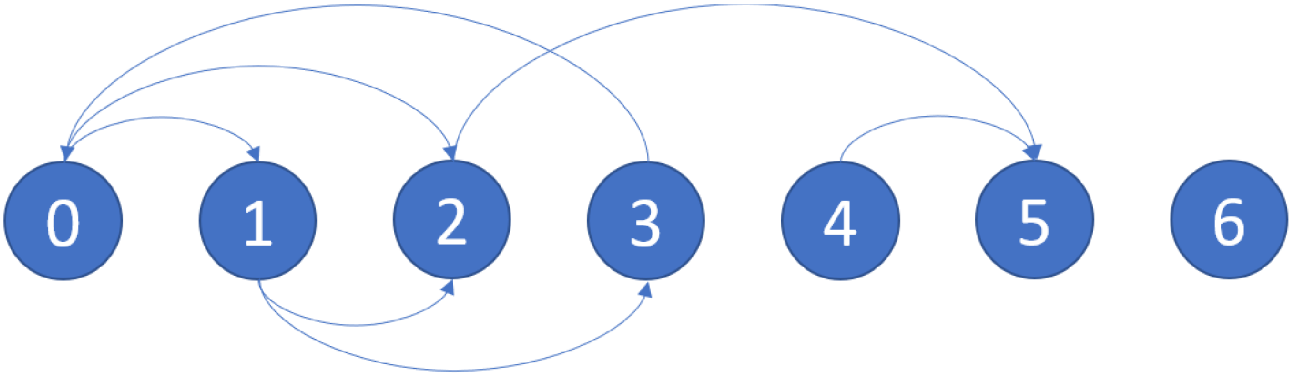
要求：找出图中所有的安全节点，将其存入数组作为答案返回，答案数组中的元素应当按升序排列。

说明：

- **终端节点**：如果一个节点没有连出的有向边，则它是终端节点。或者说，如果没有出边，则节点为终端节点。
- **安全节点**：如果从该节点开始的所有可能路径都通向终端节点，则该节点为安全节点。
- $n == graph.length$。
- $1 \le n \le 10^4$。
- $0 \le graph[i].length \le n$。
- $0 \le graph[i][j] \le n - 1$。
- $graph[i]$ 按严格递增顺序排列。
- 图中可能包含自环。
- 图中边的数目在范围 $[1, 4 \times 10^4]$ 内。

示例：

- 示例 1：




```py
输入：graph = [[1,2],[2,3],[5],[0],[5],[],[]]
输出：[2,4,5,6]
解释：示意图如上。
节点 5 和节点 6 是终端节点，因为它们都没有出边。
从节点 2、4、5 和 6 开始的所有路径都指向节点 5 或 6。
```

- 示例 2：

```py
输入：graph = [[1,2,3,4],[1,2],[3,4],[0,4],[]]
输出：[4]
解释：
只有节点 4 是终端节点，从节点 4 开     所有路径都通向节点 4。
```

解题思路

思路 1：拓扑排序

1. 根据题意可知，安全节点所对应的终点，一定是出度为 0 的节点。而安全节点一定能在有限步内到达终点，则说明安全节点一定不在「环」内。
2. 我们可以利用拓扑排序来判断顶点是否在环中。
3. 为了找出安全节点，可以采取逆序建图的方式，将所有边进行反向。这样出度为 0 的终点就变为了入度为 0 的点。
4. 然后通过拓扑排序不断移除入度为 0 的点之后，如果不在「环」中的点，最后入度一定为 0，这些点也就是安全节点。而在「环」中的点，最后入度一定不为 0。
5. 最后将所有安全的起始节点存入数组作为答案返回。

思路 1：代码

```py
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 用于记录所有节点入度
        for u in graph:
            for v in graph[u]:
                indegrees[v] += 1           # 统计所有节点入度

        # 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])

        while S:
            u = S.pop()                     # 从集合中选择一个没有前驱的顶点 0
            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

        res = []
        for u in indegrees:
            if indegrees[u] == 0:
                res.append(u)

        return res

    def eventualSafeNodes(self, graph: List[List[int]]) -> List[int]:
        graph_dict = {u: [] for u in range(len(graph))}

        for u in range(len(graph)):
            for v in graph[u]:
                graph_dict[v].append(u)     # 逆序建图

        return self.topologicalSortingKahn(graph_dict)
```

思路 1：复杂度分析

- **时间复杂度**：$O(n+m)$，其中 n 是图中节点数目，m 是图中边数目。
- **空间复杂度**：$O(n+m)$。