问题：N 个节点的工作流(有向无环图)中，找到所有从 0 到(N-1)的路径

输入：二维数组，第 i 个数组中表示有向图中 i 号结点所能到达的下个节点

输出：所有的路径

代码：采用深度优先遍历

```
class Solution:
    def __init__(self):
        self.res = []

    # 输入: (track, graphs, dot, out_degree)
    # track: 每轮递归所产生的路径
    # graphs: 邻接矩阵
    # dot: 每轮递归当前的点的下标
    # out_degree: 所有点的出度
    def backtrack(self, track, graphs, dot, out_degree):
        # 当出现出度为 0 的点，则将结果写入 track，结束递归
        if len(track) != 0 and out_degree[dot] == 0:
            self.res.append(track[:])  # 深拷贝
            return None

        for i in range(0, len(graphs[dot])):
            if graphs[dot][i] != 0:
                if i not in track:
                    track.append(i)
                    self.backtrack(track, graphs, i, out_degree)
                    track.pop(-1)

    def find_track(self, graph):
        graphs = [[0 for j in range(len(graph))] for i in range(len(graph))]

        in_degree = [0] * len(graph)  # 记录所有点的入度
        out_degree = [0] * len(graph)  # 记录所有点的出度
        for index, dots in enumerate(graph):
            if len(dots) != 0:
                for dot in dots:
                    graphs[index][dot] = 1  # 邻接表
                    in_degree[dot] += 1
            out_degree[index] = len(dots)

```

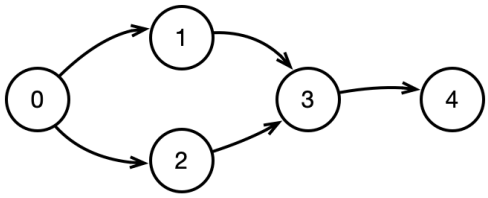
```
        for i in range(len(in_degree)):
            # 输入入度为 0 的点
            if in_degree[i] == 0:
                self.backtrack([i], graphs, 0, out_degree)

        return self.res


solution = Solution()
print(solution.find_track([[2], [2], [3, 4], [], []]))
```

用例 1：

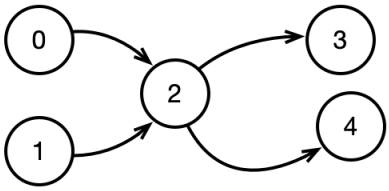
输入：[[1,2],[3],[3],[4],[]]



输出：[[0, 1, 3, 4], [0, 2, 3, 4]]

用例 2：

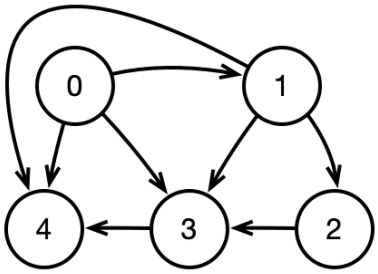
输入：[[2], [2], [3, 4], [], []]



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

用例 3：

输入：[[4,3,1],[3,2,4],[3],[4],[]]



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