这篇文章旨在说明如何使用DFS或BFS求解有向图的拓扑排序，如果存在环则拓扑排序不存在。

**BFS**：BFS特点为从一个点开始，记录其所有的邻接点之后才会进入下一个点继续遍历。而对于拓扑排序来说，一个点的入度决定了这个点在接下来的排序中是否可以作为其余点的前序点，因为只有入度为0的点才可以作为后面序列的起始点，而入度>0的点则表示在他之前还有要依赖的点。以上课为例，上课程A需要先学B，那么A入度为1，只有在B学完之后A入度变为0，此时才可以学习A。通过在循环过程中，将入度为0的点加入到队列中，如果最终存在拓扑排序则可以得到一个序列，这个序列是入度减为0的先后顺序，如果最终没有访问到所有点，则存在环，因为环中的入度始终大于1，就没有入度=0的点也就无法打开缺口，这些点也始终无法进入队列进行后续遍历。

代码;

edge表示有向边，in\_degree表示每个点的入度，q表示循环的队列，ans记录拓扑排序的结果（多个任意返回其一）

while q:

node = q.popleft()

ans.append(node)

for e in edge[node]:

in\_degree[e] -= 1

if in\_degree[e] == 0:

q.push(e)

return ans #需要判断是否每个点都访问过

**DFS:**DFS特点是对一条路径走到底，无法继续往前时停止并回溯。这个特点刚好和点的出度相符，如果点的出度为0，则表示没有课程需要在完成他之后才能完成，那么这个点可以作为一个拓扑序列的终点，或者说对其他点没有影响的点。而DFS使用递归调用栈的形式，也和点的访问顺序相符，先放入对其余点没有影响的点，那么这个点在真正的拓扑排序中一定时靠后的，所以按照栈的顺序弹出（即逆序）即可得到拓扑排序。做法是，标记每个点为未搜索，搜索中和已搜索，如果一个点的出度为0，则该点搜索完成，放入栈中。如果搜索过程中遇到了邻接点是搜索中，则一定存在环，因为无法找到一个出度为0的更靠后的点。如果该点是未搜索，则按照递归顺序对所有邻接点进行深搜。

代码：

edge表示有向边，out\_degree表示每个点的出度，ans记录拓扑排序的结果，0，1，2分别表示未搜索、搜索中和已搜索，记录在visited数组中。

def dfs(node):

if len(edge[node]) == 0:

return

visited[node] = 1

for x in edge[node]:

if visited[x] == 1:

cycle\_exist = True

return

elif visited[x] == 0:

dfs(x)

visited[node] = 2

ans.append(node)