```
def allPathsSourceTarget(graph):
    #graph = [[1,2],[3],[3],[]]
    output = []
    def get_paths(node, graph, path, target):
        if node == target:
            output.append(path+[node])
        else:
            for item in graph[node]:
                get_paths(item, graph, path+[node], target)
    source, target = 0, len(graph)-1
    get_paths(source, graph, [], target)
    return output
```

```
# Time complexity : O(n). We need to traverse over the ppid array of size
n once.
# Space complexity : O(n). size of the map
def killProcessBFS(pid, ppid, kill):
        graph = defaultdict(set)
        for i in range(len(pid)):
            graph[ppid[i]].add(pid[i])
        queue= [kill]
        result = []
        while queue:
            node = queue.pop()
            res.append(node)
            for children in graph[node]:
                queue.append(children)
        return result

# Time complexity : O(n). We need to traverse over the ppid array of size
n once.
# Space complexity : O(n). size of the map
def killProcessDFS(pid, ppid, kill):
        graph = defaultdict(set)
        for i in range(len(pid)):
            graph[ppid[i]].add(pid[i])
        ans = []
        ########################################
        def dfs(node):
            ans.append(node)
            if not node:  # or not in graph
                return
            for c in graph[node]:
                dfs(c)
         ##############################################
        dfs(kill)
        return ans
```

```
# Space Complexity:O(N*2^N)
# Time Complexity: O(N*2^N)
def allPathsSourceTarget(graph):
    def dfs(node,path):
        if node == len(graph)-1: # if my node reachs my target, because
this is acyclic
            output.append(path)
        for nextNodeRef in graph[node]:
            dfs(nextNodeRef,path+[nextNodeRef])
    output=[]
    dfs(0,[0])
    return output
##########################################################################
###############
def allPathsSourceTargetStack(graph):
    result = []
    target = len(graph) - 1
    stack = [([0], graph[0])]

    while stack:
        for _ in range(len(stack)):
            path, nodes = stack.pop()

            if path and path[-1] == target:
                result.append(path)

            for neighbor in nodes:
                stack.append((path + [neighbor], graph[neighbor]))
    return result
##########################################################################
####
def allPathsSourceTargetWITHCYCLE(graph):
    seen=set()
    def dfs(node,path):
        seen.add(node)

        if node == len(graph)-1:
            output.append(path)
        for nextNodeRef in graph[node]:
            if nextNodeRef not in seen:
              dfs(nextNodeRef,path+[nextNodeRef])
            else:
              continue
    output=[]
    dfs(0,[0])
    return output
### UnlockRooms ##################################################
def solve(rooms):
    seen=set()

    def dfs(node, connections):
        seen.add(node)
```

```
        for nextRoom in rooms[node]:

            if nextRoom not in seen:
                dfs(nextRoom, connections+[nextRoom])
            else:
                continue
    dfs(0,[1,3])
    return len(seen)==len(rooms)
```