#case 1 ie when street number 6 is nearest street to the fireplace  
print("CASE 1:")  
from collections import defaultdict  
class Graph:  
 def \_\_init\_\_(self,vertices):  
 self.V= vertices  
 self.graph = defaultdict(list)  
 def addvertices(self,u,v):  
 self.graph[u].append(v)  
 def printAllPathsUtil(self, u, d, visited, path): #marking the current vertex as visited  
 visited[u]= True  
 path.append(u)  
 if u==d:  
 print(path)  
 else:  
 for i in self.graph[u]: #to remove the current vertex from the path amd we mark it as unvisited  
 if visited[i]==False:  
 self.printAllPathsUtil(i, d, visited, path)  
 path.pop()  
 visited[u]= False  
 def printAllPaths(self,s, d): # s is the start point and d is the destination point  
 visited =[False]\*(self.V)  
 path = []  
 self.printAllPathsUtil(s, d,visited, path) #to print all the paths  
#creating a graph  
g = Graph(8) #total 8 paths are given in the input  
g.addvertices(1,2)  
g.addvertices(1,3)  
g.addvertices(3,4)  
g.addvertices(3,5)  
g.addvertices(4,6)  
g.addvertices(5,6)  
g.addvertices(2,3)  
g.addvertices(3,4)  
g.addvertices(0,0)  
s = 1 ; d = 6  
print ("These are the routes possible from the firestation to the streetcorner 6:")  
g.printAllPaths(s, d)  
#case 2 when street corner 4 is nearer to the fireplace  
print("CASE 2:")  
print("These are the routes possible from the firestation to the streetcorner 4:")  
graph = { 1: [6, 8],  
 2: [3, 5],  
 3: [4,1],  
 4: [6],  
 5: [1, 7],  
 6: [9],  
 7: [8],  
 8: [9],  
 }  
def find\_all\_routes(graph, start, end, route=[]):  
 route = route + [start]  
 if start == end:  
 return [route]  
 routes = []  
 for street in graph[start]:  
 if street not in route:  
 newroutes = find\_all\_routes(graph, street, end, route)  
 for newroute in newroutes:  
 routes.append(newroute)  
 return routes  
print(find\_all\_routes(graph, 1, 9))