#DFS

tree = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F', 'G'], 'D': [], 'E': [], 'F': [], 'G': []}

start = input("Enter start node: ").strip().upper()

def dfs\_traversal(tree):

Open = [start]

close = []

while Open:

node = Open.pop()

if node not in close:

close.append(node)

neighbour = tree[node]

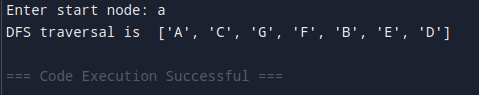
for i in neighbour:

Open.append(i)

return close

print("DFS traversal is ", dfs\_traversal(tree))

OUTPUT:



# Goal Search

tree = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F', 'G'], 'D': [], 'E': [], 'F': [], 'G': []}

start = input("Enter start node: ").strip().upper()

goal = input("Enter the goal node: ").strip().upper()

def dfs\_traversal(tree):

Open = [start]

close = []

if start == goal:

print("Start node is the goal node")

return start

while Open:

node = Open.pop()

if node not in close:

close.append(node)

neighbour = tree[node]

for i in neighbour:

Open.append(i)

if i == goal:

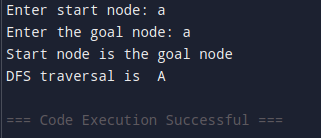
close.append(i)

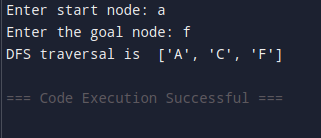
return close

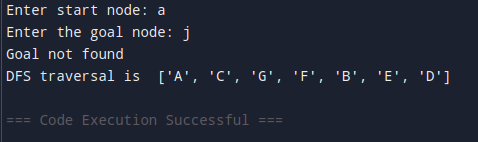
print("Goal not found")

return close

print("DFS traversal is ", dfs\_traversal(tree))

OUTPUT:  






#List of Lists

tree = {'A' : ['B','C'], 'B':['D','E'], 'C':['F','G'], 'D':[], 'E':[], 'F':[], 'G':[]}

start = input("Enter Start Node: ").strip().upper()

goal = input("Enter Goal Node: ").strip().upper()

def dfs\_absolute(tree):

open = [[start]] # to store list inside list

close = []

if start == goal:

print("Start node is the goal node")

return [start]

while open:

path = open.pop()

node = path [-1] #access last element

if node not in close:

close.append(node)

neighbor = tree [node]

for i in neighbor:

new\_path = list (path)

new\_path.append (i) #[(A) +B, (A) +C]

open.append(new\_path)

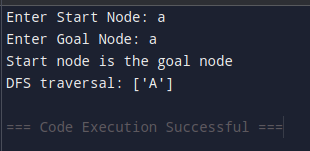
if i == goal:

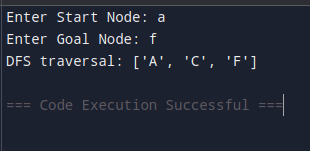
return new\_path

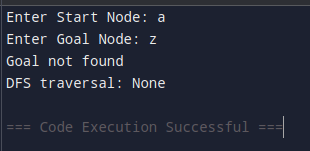
print("Goal not found")

print("DFS traversal:",dfs\_absolute(tree))

OUTPUT:







#Depth Limited DFS

tree = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F', 'G'], 'D': [], 'E': [], 'F': [], 'G': []}

start = input("Enter start node: ").strip().upper()

goal = input("Enter the goal state: ").strip().upper()

depth\_limit = int(input("Enter the depth limit: "))

path=[] # to store final result

level = 0 #to store current level

def dldfs(start,goal,tree,level,path,depth\_limit):

print("Current level is",level)

path.append(start)

if start == goal:

print("Search ends")

return path

if level == depth\_limit:

return False

print("Expanding current node",start)

neighbour = tree[start]

#To explore the neighbours

for i in neighbour:

if dldfs(i,goal,tree,level+1,path,depth\_limit):

return True

path.pop() #To find the shortest path dropping the one which will lead to a longer path\

return False

result = dldfs(start,goal,tree,level,path,depth\_limit)

if result:

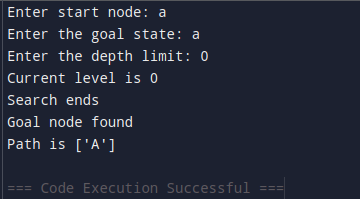
print("Goal node found")

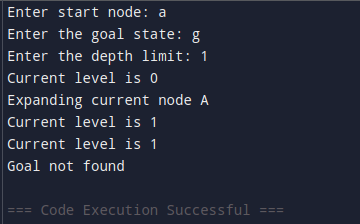
print("Path is",path)

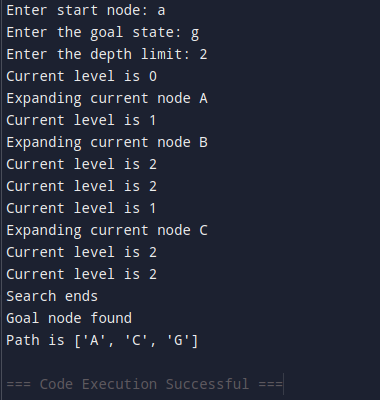
else:

print("Goal not found")

Output:







#Iterative deepening DFS

tree = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F', 'G'], 'D': [], 'E': [], 'F': [], 'G': []}

start = input("Enter start node: ").strip().upper()

goal = input("Enter the goal state: ").strip().upper()

MaxD = int(input("Enter the maximum depth of the tree: "))

path=[] # to store final result

level = 0 #to store current level

def dldfs(start,goal,tree,level,path,depth\_limit):

print("Current level is",level)

path.append(start)

if start == goal:

print("Search ends")

return path

if level == depth\_limit:

return False

print("Expanding current node",start)

neighbour = tree[start]

#To explore the neighbours

for i in neighbour:

if dldfs(i,goal,tree,level+1,path,depth\_limit):

return True

path.pop() #To find the shortest path dropping the one which will lead to a longer path

return False

def iddfs(start,goal,tree,MaxD):

for i in range(MaxD):

print("Iteration",i)

path=[]

if dldfs(start, goal, tree, level, path,i): #passing i bcoz at each depth we have to run the depth limited dfs code (i is passed as depth limit)

print("Goal Exists")

print("Path",path)

return True

else:

print("Goal not found")

return False

iddfs(start,goal,tree,MaxD)

OUTPUT:

