***EXPERIMENT 02***

1. ***Depth First Search Traversal***

def dfs(start,tree):  
    visited = []  
    stack = [start]  
    while stack:  
        node = stack.pop()  
        visited.append(node)  
          
        for adjnode in tree[node]:  
              
            if adjnode not in visited:  
                  stack.append(adjnode)  
    print("DFS traversal is: ", visited)  
tree={  
        
      'A':['B','C'],  
      'B':['D','E'],  
      'C':['F','G'],  
      'D':[],  
      'E':[],  
      'F':[],  
      'G':[]  
      }  
start = input("Enter the start node ")   
dfs(start, tree)  # T R A V E R S A  L

***Output:***

*Enter the start node A*

*DFS traversal is: ['A', 'C', 'G', 'F', 'B', 'E', 'D']*

1. ***Find goal node using Depth First Search***

visited=set()  
result=[]  
found=False  
def dfs\_search(tree,node,goal):  
    if node not in visited:  
        result.append(node)  
        visited.add(node)  
        for adjnode in tree[node]:  
               if goal in visited:  
                    global found  
                    found=True  
                    break  
               else:  
                    dfs\_search(tree,adjnode,goal)  
tree = {  
    'A': ['B', 'C'],  
    'B': ['D', 'E'],  
    'C': ['F', 'G'],  
    'D': [],  
    'E': [],  
    'F': [],  
    'G': []  
}  
start, goal = input("Enter start and goal state: ").split(" ")  
dfs\_search(tree,start,goal)  
if found:  
     print("Node found , path is: ",result)  
else:  
     print("Goal node not found")

***Output:***

*Enter start and goal state: A G*

*Node found , path is: ['A', 'B', 'D', 'E', 'C', 'F', 'G']*

1. ***Find optimal path to find goal node using Depth First Search***

def dfs\_path\_optimised(tree, start, goal):  
    visited = []  
    stack = [[start]]  
    if start == goal:  
        print(start, " is the goal node ")  
    while stack:  
        path = stack.pop()  
        node = path[-1]  
        if node not in visited:  
            visited.append(node)  
        neighbour = tree[node]  
        for i in neighbour:  
            new\_path = list(path)  
            new\_path.append(i)  
            stack.append(new\_path)  
            if i == goal:  
                return new\_path  
tree = {  
    'A': ['B', 'C'],  
    'B': ['D', 'E'],  
    'C': ['F', 'G'],  
    'D': [],  
    'E': [],  
    'F': [],  
    'G': []  
}  
  
start, goal = input("Enter start and goal state: ").split(" ")  
print("Shortest Path is : ", dfs\_path\_optimised(tree, start, goal))

***Output:***

*Enter start and goal state: A G*

*Shortest Path is : ['A', 'C', 'G']*

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

1. ***Depth Limited Depth First Search***

def Dedpth\_limit\_srch(tree, start, goal, depth\_limit, level, path):  
    print("Current level : ", level)  
    if start is goal:  
        print("Node found -> path is: ", path)  
        return True  
    else:  
        print(f"EXPANDING NODE: {start} at level : {level}")  
    if level == depth\_limit:  
        print(f"{start} is at level > DEPTH LIMIT: {depth\_limit} HENCE \_STOP\_")  
        return False  
    neighbor = tree[start]  
    for node in neighbor:  
        path.append(node)  
        if Dedpth\_limit\_srch(tree, node, goal, depth\_limit, level+1, path):  
            return True  
        path.pop()  
    return False  
tree = {  
    'A': ['B', 'C'],  
    'B': ['D', 'E'],  # B LEVEL = 1  
    'C': ['F', 'G'],  # C LEVEL = 1  
    'D': [],  
    'E': [],  
    'F': [],  
    'G': []  
}  
start, goal, depth\_limit = input(  
    "Enter styart state, goal state anbd maximum depth: ").split(" ")  
depth\_limit = int(depth\_limit)  
path = [start]  
if Dedpth\_limit\_srch(tree, start, goal, depth\_limit, 0, path) is False:  
    print(f"{goal} not found")

***Output:***

*Enter styart state, goal state anbd maximum depth: A G 2*

*Current level : 0*

*EXPANDING NODE: A at level : 0*

*Current level : 1*

*EXPANDING NODE: B at level : 1*

*Current level : 2*

*EXPANDING NODE: D at level : 2*

*D is at level > DEPTH LIMIT: 2 HENCE \_STOP\_*

*Current level : 2*

*EXPANDING NODE: E at level : 2*

*E is at level > DEPTH LIMIT: 2 HENCE \_STOP\_*

*Current level : 1*

*EXPANDING NODE: C at level : 1*

*Current level : 2*

*EXPANDING NODE: F at level : 2*

*F is at level > DEPTH LIMIT: 2 HENCE \_STOP\_*

*Current level : 2*

*Node found -> path is: ['A', 'C', 'G']*

1. ***Iterative Depth Limited Depth First Search***

def Dedpth\_limit\_srch(tree, start, goal, depth\_limit, level, path):  
    print("Current level : ", level)  
    if start is goal:  
        print("Node found -> path is: ", path)  
        return True  
    else:  
        print(f"EXPANDING NODE: {start} at level : {level}")  
    if level == depth\_limit:  
        print(f"{start} is at level > DEPTH LIMIT: {depth\_limit} HENCE \_STOP\_")  
        return False  
    neighbor = tree[start]  
    for node in neighbor:  
        # If node is found true will be returned => if so stop by returning  
        # Othwise continue with opther neighbors  
        path.append(node)  
        if Dedpth\_limit\_srch(tree, node, goal, depth\_limit, level+1, path):  
            return True  
        path.pop()  
    return False  
tree = {  
    'A': ['B', 'C'],  
    'B': ['D', 'E'],  # B LEVEL = 1  
    'C': ['F', 'G'],  # C LEVEL = 1  
    'D': [],  
    'E': [],  
    'F': [],  
    'G': []  
}  
start, goal = input("Enter start state, goal state: ").split(" ")  
path = [start]  
max\_depth = 2  
print(f"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
for depth\_limit in range(0, max\_depth+1):  
    print(f"\_\_\_\_\_\_\_\_\_\_\_\_\_\_MAXIMUM DEPTH: {depth\_limit}\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
    path = [start]  
    if Dedpth\_limit\_srch(tree, start, goal, depth\_limit, 0, path) is False:  
        print(f"{goal} not found")  
    print(f"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

***Output:***

***Enter start state, goal state: A G***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_MAXIMUM DEPTH: 0\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Current level : 0***

***EXPANDING NODE: A at level : 0***

***A is at level > DEPTH LIMIT: 0 HENCE \_STOP\_***

***G not found***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_MAXIMUM DEPTH: 1\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Current level : 0***

***EXPANDING NODE: A at level : 0***

***Current level : 1***

***EXPANDING NODE: B at level : 1***

***B is at level > DEPTH LIMIT: 1 HENCE \_STOP\_***

***Current level : 1***

***EXPANDING NODE: C at level : 1***

***C is at level > DEPTH LIMIT: 1 HENCE \_STOP\_***

***G not found***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_MAXIMUM DEPTH: 2\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Current level : 0***

***EXPANDING NODE: A at level : 0***

***Current level : 1***

***EXPANDING NODE: B at level : 1***

***Current level : 2***

***EXPANDING NODE: D at level : 2***

***D is at level > DEPTH LIMIT: 2 HENCE \_STOP\_***

***Current level : 2***

***EXPANDING NODE: E at level : 2***

***E is at level > DEPTH LIMIT: 2 HENCE \_STOP\_***

***Current level : 1***

***EXPANDING NODE: C at level : 1***

***Current level : 2***

***EXPANDING NODE: F at level : 2***

***F is at level > DEPTH LIMIT: 2 HENCE \_STOP\_***

***Current level : 2***

***Node found -> path is: ['A', 'C', 'G']***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***