PROGRAM 01 : Depth First Search Traversal

```
graph = {
    'A' : ['B', 'C'],
    'B' : ['D', 'E'],
    'C' : ['F', 'G'],
    'D' : [],
    'E' : [],
    'F' : [],
    'G' : []
}
start = input("Enter the start node : ")
def depthFirstSearch(graph):
    visited = []
    stack = [start]

    while stack:
        node = stack.pop()
        if node not in visited:
            visited.append(node)
            neighbours = graph[node]
            for neighbour in neighbours:
                stack.append(neighbour)
    return visited

print(f"DFS Traversal : {depthFirstSearch(graph)}")
```

OUTPUT :

```
Enter the start node : A
DFS Traversal : ['A', 'C', 'G', 'F', 'B', 'E', 'D']
```

PROGRAM 02 : DFS Goal Search

```
graph = {
    'A' : ['B', 'C'],
    'B' : ['D', 'E'],
    'C' : ['F', 'G'],
    'D' : [],
    'E' : [],
    'F' : [],
    'G' : []
}
visited = []
start = input("Enter the start node : ")
goal = input("Enter the goal node  : ")

def goalDepthFirstSearch(graph, start):
    if start not in visited:
        visited.append(start)
        neighbours = graph[start]

        for neighbour in neighbours:
            if goal in visited:
                print(visited)
```

```
                break
            else:
                goalDepthFirstSearch(graph, neighbour)

print("\nDFS Traversal to goal node : ")
goalDepthFirstSearch(graph, start)
```

OUTPUT :

```
Enter the start node : A
Enter the goal node  : F

DFS Traversal to goal node :
['A', 'B', 'D', 'E', 'C', 'F']
```

```
Enter the start node : A
Enter the goal node  : A

DFS Traversal to goal node :
['A']
```

PROGRAM 03 : DFS Shortest Path

```
graph = {
    'A' : ['B', 'C'],
    'B' : ['D', 'E'],
    'C' : ['F', 'G'],
    'D' : [],
    'E' : [],
    'F' : [],
    'G' : []}
start = input("Enter start node : ")
goal = input("Enter goal node  : ")

def dfsShortestPath(graph):
    visited = []
    stack = [[start]]
    while stack:
        path = stack.pop()
        node = path[-1]
        if node not in visited:
            visited.append(node)
            neighbours = graph[node]
            for neighbour in neighbours:
                new_path = list(path)
                new_path.append(neighbour)
                stack.append(new_path)
                if neighbour == goal:
                    return new_path
print(f"Shortest DFS Path is : {dfsShortestPath(graph)}")
```

OUTPUT :

```
Enter start node : A
Enter goal node  : F
Shortest DFS Path is : ['A', 'C', 'F']
```

PROGRAM 04 : Depth Limit DFS

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': [],
    'E': [],
    'F': [],
    'G': []
}
start = input("Enter start node    : ")
goal = input("Enter the goal node : ")
maxlim = int(input("Enter the max limit : "))
level = 0
path = []

def depth_limit_dfs(start, goal, graph, path, level, maxlim):
    print("Current level : ", level)
    path.append(start)
    if start == goal:
        print("goal found")
        return path
    if level == maxlim:
        print("Maximum Level Reached")
        return False
    print("Expanding node: ",start)
    neighbours = graph[start]
    for neighbour in neighbours:
        if depth_limit_dfs(neighbour, goal, graph, path, level+1, maxlim):
            return path
        path.pop()
    return False

if depth_limit_dfs(start, goal, graph, path, level, maxlim):
    print("Goal id found")
    print(f"The shortest path is {path}")
else:
    print("Goal not found")
```

OUTPUT :

```
Enter start node    : A
Enter the goal node : F
Enter the max limit : 2
Current level :  0
Expanding node:  A
Current level :  1
Expanding node:  B
Current level :  2
Maximum Level Reached
Current level :  2
Maximum Level Reached
Current level :  1
Expanding node:  C
Current level :  2
goal found
Goal id found
The shortest path is ['A', 'C', 'F']
```

PROGRAM 05 : Iterative Depth Limit DFS

```
graph = {
    'A' : ['B', 'C'],
    'B' : ['D', 'E'],
    'C' : ['F', 'G'],
    'D' : [],
    'E' : [],
    'F' : [],
    'G' : []
}
start = input("Enter start node    : ")
goal = input("Enter the goal node : ")
level = 0
path = []
maxiteration = 100

def dldfs(start, goal, graph, path, level, maxlim):
    print("Current level : ", level)
    path.append(start)
    if start == goal:
        print("goal found")
        return path
    if level == maxlim:
        print("Maximum Level Reached")
        return False
    print("expanding node: ",start)
    neighbour = graph[start]
    for i in neighbour:
```

```
        if dldfs(i, goal, graph, path, level+1, maxlim):
            return path
        path.pop()
    return False

def iterativedldfs(start, goal, graph, maxiteration):
    for i in range(maxiteration):
        path = []
        print("iteration: ", i+1)
        if dldfs(start, goal, graph, path, level, i):
            print("goal found")
            print("path: ", path)
            return True
    return False

iterativedldfs(start, goal, graph, maxiteration)
```

OUTPUT :

```
Enter start node    : A
Enter the goal node : F
iteration:  1
Current level :  0
Maximum Level Reached
iteration:  2
Current level :  0
expanding node:  A
Current level :  1
Maximum Level Reached
Current level :  1
Maximum Level Reached
iteration:  3
Current level :  0
expanding node:  A
Current level :  1
expanding node:  B
Current level :  2
Maximum Level Reached
Current level :  2
Maximum Level Reached
Current level :  1
expanding node:  C
Current level :  2
goal found
goal found
path:  ['A', 'C', 'F']
```