Code Implementation:

Answer(1):

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

}
print(tree)
```

Output:

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

Answer(2):

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

def dfs(graph, start, goal):
    visited = set()
    parent = {}

    def dfs_recursive(node):
        if node == goal:
```

Lab Task:


```
            return True

        visited.add(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                parent[neighbor] = node
                if dfs_recursive(neighbor):
                    return True
        return False

    if dfs_recursive(start):
        return reconstruct_path(parent, start, goal)
    else:
        return None

def reconstruct_path(parent, start, goal):
    path = []
    current = goal
    while current != start:
        path.append(current)
        current = parent[current]
    path.append(start)
    return path[::-1]
start = 'A'
goal = 'G'
path = dfs(tree, start, goal)
if path:
    print("Path found:", path)
else:
    print("No path found.")
```

Output:

Path found: ['A', 'C', 'G']