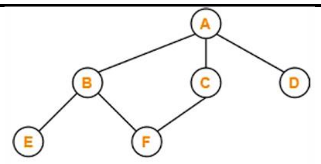
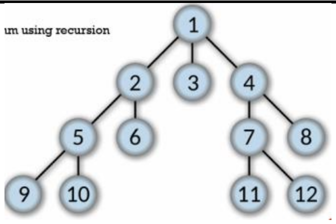
Bahria University,

Karachi Campus

LAB EXPERIMENT NO.
_01_

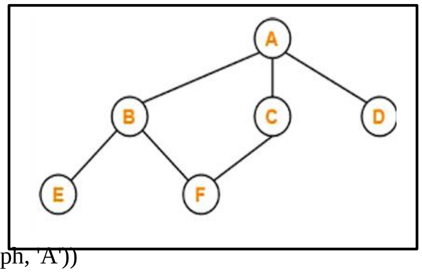
LIST OF TASKS

TASK NO	OBJECTIVE
1	Implement BFS & DFS Algorithm in python on the given graph:
	
2	Implement the BFS and DFS Algorithm using recursion on the given graph:
	 um using recursion 1 2 3 4 5 6 7 8 9 10 11 12
3	Apply the UCS algorithm on a map given below. Find optimal cost from ARAD to BUCHAREST.

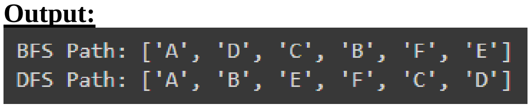
Submitted On:
Date: _____4/5/2024______

Task No.01:Implement BFS & DFS Algorithm in python on the given graph.

```
def bfs(graph, start):
    visited = set()
    queue = [start]
    traversal = []
    while queue:
        vertex = queue.pop(0)
        if vertex not in visited:
            visited.add(vertex)
            traversal.append(vertex)
            queue.extend(graph[vertex] - visited)
    return traversal
def dfs(graph, start):
    visited = set()
    stack = [start]
    traversal = []
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            traversal.append(vertex)
            stack.extend(graph[vertex] - visited)
    return traversal
graph = {
    'A': set(['B', 'C', 'D']),
    'B': set(['E', 'F']),
    'C': set(['F']),
    'D': set([]),
    'E': set([]),
    'F': set([])
}
print("BFS Path:", bfs(graph, 'A'))
print("DFS Path:", dfs(graph, 'A'))
```



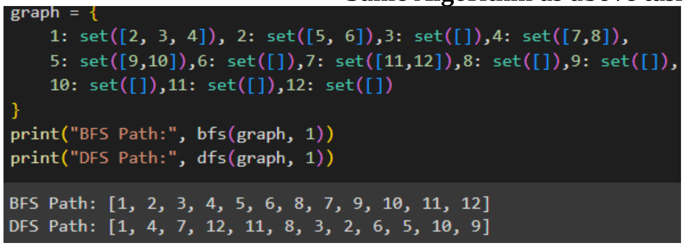
Output:



```
BFS Path: ['A', 'D', 'C', 'B', 'F', 'E']
DFS Path: ['A', 'B', 'E', 'F', 'C', 'D']
```

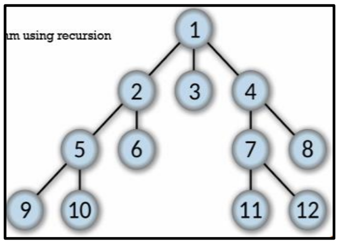
Task No.02: Implement the BFS and DFS Algorithm using recursion on the given graph:

Same Algorithm as above task:



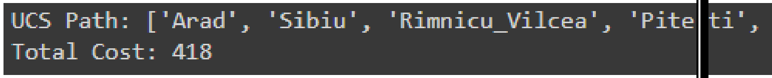
```
graph = {
    1: set([2, 3, 4]), 2: set([5, 6]),3: set([]),4: set([7,8]),
    5: set([9,10]),6: set([]),7: set([11,12]),8: set([]),9: set([]),
    10: set([]),11: set([]),12: set([])
}
print("BFS Path:", bfs(graph, 1))
print("DFS Path:", dfs(graph, 1))
```

```
BFS Path: [1, 2, 3, 4, 5, 6, 8, 7, 9, 10, 11, 12]
DFS Path: [1, 4, 7, 12, 11, 8, 3, 2, 6, 5, 10, 9]
```



<u>Task No.03:</u> Apply the UCS algorithm on a map given below. Find optimal cost from ARAD to BUCHAREST.

```
import heapq
def ucs(graph, start, goal):
    visited = set()
    queue = [(0, start)]
    path = {}
    while queue:
        cost, vertex = heapq.heappop(queue)
        if vertex not in visited:
            visited.add(vertex)
            if vertex == goal:
                final_path = []
                while vertex in path:
                    final_path.append(vertex)
                    vertex = path[vertex]
                final_path.append(start)
                return final_path[::-1], cost
            for neighbor, neighbor_cost in graph[vertex].items():
                if neighbor not in visited:
                    total_cost = cost + neighbor_cost
                    heapq.heappush(queue, (total_cost, neighbor))
                    if neighbor not in path or total_cost < graph[vertex][neighbor]:
                        path[neighbor] = vertex
    return None, float('inf')
graph = {
    'Arad': {'Zerind': 75, 'Timisoara': 118, 'Sibiu': 140},
    'Zerind': {'Oradea': 71},
    'Bucharest': {'Fagaras': 211, 'Giurgiu': 90, 'Pitesti': 101, 'Urziceni': 85},
    'Craiova': {'Dobreta': 120, 'Pitesti': 138, 'Rimnicu_Vilcea': 146},
    'Dobreta': {'Mehadia': 75},
    'Eforie': {'Hirsova': 86},
    'Fagaras': {'Sibiu': 99},
    'Hirsova': {'Urziceni': 98},
    'Iasi': {'Neamt': 87, 'Vaslui': 92},
    'Lugoj': {'Mehadia': 70, 'Timisoara': 111},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Pitesti': {'Rimnicu_Vilcea': 97},
    'Rimnicu_Vilcea': {'Sibiu': 80},
    'Urziceni': {'Vaslui': 142}
}
start = 'Arad';oal = 'Bucharest'
path, cost = ucs(graph, start, goal)
if path:
    print("UCS Path:", path)
    print("Total Cost:", cost)
else:
    print("No path found from", start, "to", goal)
```



```
UCS Path: ['Arad', 'Sibiu', 'Rimnicu_Vilcea', 'Pitesti',
Total Cost: 418
```