Experiment 03

UCS:

```
import heapq

def uniform_cost_search(graph, start, goal):
        heap = [(0, start, [])]
        visited = set()
        print("Visted Nodes:",end="")

        while heap:
        cost, node, path = heapq.heappop(heap)
        if node not in visited:
        print(node,end="")
        visited.add(node)
        path = path + [node]
        if node == goal:
                return path, cost
        for neighbor, edge_cost in graph.get(node, []):
                if neighbor not in visited:
                heapq.heappush(heap, (cost + edge_cost, neighbor, path))
        return None, float('inf')

graph = {
        'A': [('B', 1), ('C', 4)],
        'B': [('D', 2), ('E', 5)],
        'C': [('F', 3)],
        'D': [],
        'E': [],
        'F': []
}

# Graph :
#       A
#       1/ \4
#       B   C
#       2/ \5 \3
#       D   E   F

start_node = 'A'
goal_node = 'F'
path, cost = uniform_cost_search(graph, start_node, goal_node)
```

```
if path:
    print()
    print("Path:", path)
    print("Cost:", cost)
else:
    print("No path found from", start_node, "to", goal_node)
```

OUTPUT:

```
Visted Nodes:ABDCEF
Path: ['A', 'C', 'F']
Cost: 7
```

BFS:

```
from collections import deque

def bfs(graph, start, goal):
    visited = set()
    queue = deque([start])
    path = {start: None}
    print("Path: ", end='')
    while queue:
    current_node = queue.popleft()
    print(current_node, end='')
    if current_node == goal:
    print()
    print("Goal found")
    return True

    visited.add(current_node)

    for neighbor in graph[current_node]:
    if neighbor not in visited and neighbor not in queue:
        queue.append(neighbor)
        path[neighbor] = current_node

    return False
#
#   A
#   / \
#       B   C
#       / \   \
#     D   E   F
graph = {
        'A': ['B', 'C'],
        'B': ['D', 'E'],
        'C': ['F'],
        'D': [],
        'E': [],
        'F': []
}

start_node = input("Enter the start node: ")
goal_node = input("Enter the goal node: ")

bfs_path = bfs(graph, start_node, goal_node)

if bfs_path:
        print(f"Goal '{goal_node}' found using BFS.")

else:
        print(f"Goal '{goal_node}' not found using BFS.")
```

OUTPUT:

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

Path: ABCDEF
Goal found
Goal 'F' found using BFS.
```