**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**

# AI & Decision Support Systems

## Lab Report #3

**Student Name: Nawab Aarij Imam**

**Degree/ Syndicate: 43 CE - A**

## Task1:
## Code:

```
    def bfs(self, start):
        visited = set()
        queue = deque([start])
        order = []

        while queue:
            vertex = queue.popleft()
            if vertex not in visited:
                visited.add(vertex)
                order.append(vertex)
                for neighbor in self.graph[vertex]:
                    if neighbor not in visited:
                        queue.append(neighbor)

        return order
graph = UndirectedGraph()

    for vertex in ['A', 'B', 'C', 'D', 'E']:
        graph.add_vertex(vertex)


    edges = [
        ('A', 'B'), ('A', 'D'), ('A', 'E'),
        ('B', 'E'), ('B', 'D'),
        ('D', 'C')
    ]

    for edge in edges:
        graph.add_edge(edge[0], edge[1])

    graph.print_graph()

    print(graph.bfs('C'))
```

**Output:**

```
❯ python task1.py
A : ['B', 'D', 'E']
B : ['A', 'E', 'D']
C : ['D']
D : ['A', 'B', 'C']
E : ['A', 'B']
['C', 'D', 'A', 'B', 'E']
```

## Task2:

## Code:

```
from collections import deque
class Tree:
    def __init__(self, data=None):
        self.data = data
        self.left = None
        self.middle = None
        self.right = None

def bfs(root):
    if root is None:
        return []

    queue = deque([root])
    order = []

    while queue:
        current_node = queue.popleft()
        order.append(current_node.data)
        if current_node.left:
            queue.append(current_node.left)
        if current_node.middle:
            queue.append(current_node.middle)
        if current_node.right:
```

```
        queue.append(current_node.right)

    return order


root = Tree(1)
root.left = Tree(2)
root.middle = Tree(3)
root.right = Tree(4)
root.left.left = Tree(5)
root.left.right = Tree(6)
root.right.left = Tree(7)
root.right.right = Tree(8)
root.left.left.left = Tree(9)
root.left.left.right = Tree(10)
root.right.left.left = Tree(11)
root.right.left.right = Tree(12)


bfs_result = bfs(root)
print("BFS Traversal Order:", bfs_result)
```

## Output:

```
❯ python task2.py
BFS Traversal Order: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

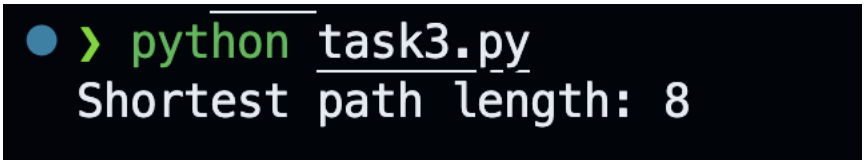
## Task3:

## Code:

```
from collections import deque

def shortest_path_bfs(maze):
    if not maze or not maze[0]:
        return -1

    rows, cols = len(maze), len(maze[0])
```

```
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    queue = deque([(0, 0, 0)])
    visited = set((0, 0))

    while queue:
        row, col, distance = queue.popleft()

        if row == rows - 1 and col == cols - 1:
            return distance

        for dr, dc in directions:
            new_row, new_col = row + dr, col + dc



            if 0 <= new_row < rows and 0 <= new_col < cols and maze[new_row][new_col] == 0 and
(new_row, new_col) not in visited:
                visited.add((new_row, new_col))
                queue.append((new_row, new_col, distance + 1))
    return -1

if __name__ == "__main__":
    maze = [
        [0, 1, 0, 0, 0],
        [0, 1, 0, 1, 0],
        [0, 0, 0, 1, 0],
        [0, 1, 1, 0, 0],
        [0, 0, 0, 0, 0]
    ]

    # Find the shortest path
    result = shortest_path_bfs(maze)
    print("Shortest path length:", result)
```

## Output:


❯ python task3.py
Shortest path length: 8