ASSIGNMENT 8 :Maze solver using BFS & DFS

```
from queue import Queue

def bfs(maze, start, end):
    queue = Queue()
    queue.put([start])  # Enqueue the start position

    while not queue.empty():
        path = queue.get()  # Dequeue the path
        x, y = path[-1]     # Current position is the last element of the path

        if (x, y) == end:
            return path  # Return the path if end is reached

        for dx, dy in [(1,0), (0,1), (-1,0), (0,-1)]:  # Possible movements
            next_x, next_y = x + dx, y + dy
            if maze[next_x][next_y] != '#' and (next_x, next_y) not in path:
                new_path = list(path)
                new_path.append((next_x, next_y))
                queue.put(new_path)  # Enqueue the new path

# Example usage
maze = [
    ['#', '#', '#', '#', '#', '#'],
    ['#', 'S', ' ', ' ', ' ', '#'],
    ['#', ' ', '#', ' ', '#', '#'],
    ['#', ' ', '#', ' ', ' ', '#'],
    ['#', ' ', ' ', ' ', 'E', '#'],
    ['#', '#', '#', '#', '#', '#']
]
start = (1, 1)  # Start position (S)
end = (4, 4)    # End position (E)
path = bfs(maze, start, end)
print(path)
```

```
def dfs(maze, start, end):
    stack = [start] # Initialize stack with start position
    visited = set() # Track visited positions

    while stack:
        position = stack.pop() # Get current position
        x, y = position

        # Check if we've reached the end
        if position == end:
            return True

        # Mark the current cell as visited
        visited.add((x, y))

        # Explore neighbors (up, down, left, right)
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            new_x, new_y = x + dx, y + dy

            # Check bounds and if the cell is already visited or is a wall
            if (0 <= new_x < len(maze) and 0 <= new_y < len(maze[0]) and
                    maze[new_x][new_y] == 0 and (new_x, new_y) not in visited):
                stack.append((new_x, new_y))

    return False # Return False if no path is found

# Example maze: 0 -> open path, 1 -> wall
maze = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]

# Start and end positions
start = (0, 0)
end = (4, 4)

# Solve the maze
print(dfs(maze, start, end)) # Output: True
```