Aashay Gandhi

700747115

axg71150@ucmo.edu

Programming Assignment 1

Input:-

```
graph = {
    'S': ['C', 'B'],
    'B': ['E', 'D'],
    'C': ['F'],
    'E': ['I','G'],
    'D': ['H'],
    'F': ['J'],
    'H': [],
    'G': [],
    'I': [],
    'J': []
}
start = 'S'
goal = 'G'
```

Exercise 1

Implement a Depth-First Search algorithm to find the path from the start node 'S' to the goal node 'G'. Please consider the topology that is shared in Figure 1.

- Your implementation should correctly navigate this graph using DFS to find a path from 'S' to 'G'.
- Your DFS algorithm should output the path from 'S' to 'G' as a sequence of nodes.
- Include comments and clear documentation in your code for clarity.
- Ensure that your code is well-structured and follows best coding practices.

```
def dfs(graph, start, goal):
    # keep track of explored nodes
    explored = []
    # keep track of all the paths to be checked
    stack = [[start]]

    # return path if start is goal
    if start == goal:
        return stack[0]

    # keeps looping until all possible paths have been checked
    while stack:
        # pop the last path from the queue
        path = stack.pop()
        # get the last node from the path
        node = path[-1]
        print(f"Traversed to node {node}: {path}")
```

```
        if node == goal: # last node is equal to the goal then
return it or else continue to find the goal
            return path
        if node not in explored: #avoid repeated looping of the
            neighbours = graph[node]
            # go through all neighbour nodes, construct a new path
and
            # push it into the queue
            for neighbour in neighbours:
                new_path = list(path)
                new_path.append(neighbour)
                stack.append(new_path)
            # mark node as explored
            explored.append(node)

    # in case there's no path between the 2 nodes
    return 'No path found for this goal.'

result = dfs(graph, start, goal)
print(f'Path from {start} to {goal} using DFS: {result}')
```

Output:-

```
Traversed to node S: ['S']
Traversed to node B: ['S', 'B']
Traversed to node D: ['S', 'B', 'D']
Traversed to node H: ['S', 'B', 'D', 'H']
Traversed to node E: ['S', 'B', 'E']
Traversed to node G: ['S', 'B', 'E', 'G']
Path from S to G using DFS: ['S', 'B', 'E', 'G']
```

Exercise 2

Implement a Breadth-First Search algorithm to find the path from the start node 'S' to the goal node 'G'. You are provided with the same graph topology as in Task 1.

- Your implementation should correctly navigate this graph using BFS to find a path from 'S' to 'G'.
- Your BFS algorithm should output the path from 'S' to 'G' as a sequence of nodes.
- Include comments and clear documentation in your code for clarity.
- Ensure that your code is well-structured and follows best coding practices.

```
def bfs(graph, start, goal):
    # keep track of explored nodes
    explored = []
    # keep track of all the paths to be checked
    queue = [[start]]

    # return path if start is goal
    if start == goal:
        return queue[0]
```

```
    # keeps looping until all possible paths have been checked
    while queue:
        # pop the first path from the queue
        path = queue.pop(0)
        # get the last node from the path
        node = path[-1]
        print(f"Traversed to node {node}: {path}")
        if node == goal: # last node is equal to the goal then
return it or else continue to find the goal
            return path
        if node not in explored:
            neighbours = graph[node]
            # go through all neighbour nodes, construct a new path
and
            # push it into the queue
            for neighbour in neighbours:
                new_path = list(path)
                new_path.append(neighbour)
                queue.append(new_path)
            # mark node as explored
            explored.append(node)

    # in case there's no path between the 2 nodes
    return 'No path found for this goal.'
result = bfs(graph, start, goal)
print(f'Path from {start} to {goal} using BFS: {result}')
```

Output:-