Usama Mehboob **Roll No 225** **Section E**

LAB # 11

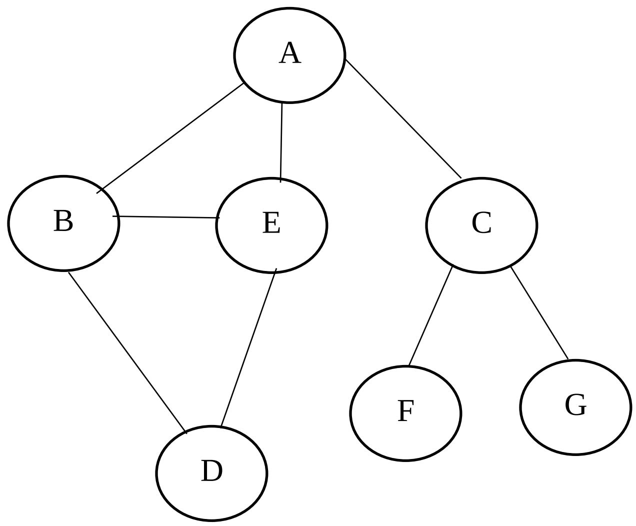
IMPLEMENTING UNINFORMED SEARCH TECHNIQUIES

OBJECTIVE

Finding shortest path using BFS and DFS search technique in python.

Lab Tasks:

1. Apply Breadth First Search on following graph considering the initial state is A and final state is G. Show results in form of open and closed list.



Code:

```
from collections import deque

def bfs(graph, start, goal):
    open_list = deque([start])
    closed_list = []
    steps = []

    while open_list:
        current = open_list.popleft()
        closed_list.append(current)


        steps.append((list(open_list), list(closed_list)))

        if current == goal:
            print("Goal reached!")
            return steps


        for neighbor in graph[current]:
            if neighbor not in open_list and neighbor not in closed_list:
                open_list.append(neighbor)

    return steps
```

```python
        for neighbor in graph[current]:
            if neighbor not in open_list and neighbor not in closed_list:
                open_list.append(neighbor)

    return steps


graph = {
    'A': ['B', 'E', 'C'],
    'B': ['A', 'E'],
    'C': ['A', 'F', 'G'],
    'E': ['A', 'B', 'D'],
    'D': ['E'],
    'F': ['C'],
    'G': ['C']
}


start = 'A'
goal = 'G'
steps = bfs(graph, start, goal)

print("Steps (Open List, Closed List):")
for i, (open_list, closed_list) in enumerate(steps, start=1):
    print(f"Step {i}: Open List = {open_list}, Closed List = {closed_list}")
```

Output:

```
Goal reached!
Steps (Open List, Closed List):
Step 1: Open List = [], Closed List = ['A']
Step 2: Open List = ['E', 'C'], Closed List = ['A', 'B']
Step 3: Open List = ['C'], Closed List = ['A', 'B', 'E']
Step 4: Open List = ['D'], Closed List = ['A', 'B', 'E', 'C']
Step 5: Open List = ['F', 'G'], Closed List = ['A', 'B', 'E', 'C', 'D']
Step 6: Open List = ['G'], Closed List = ['A', 'B', 'E', 'C', 'D', 'F']
Step 7: Open List = [], Closed List = ['A', 'B', 'E', 'C', 'D', 'F', 'G']
```

2. Using Question no. 1 apply BFS by taking initial and final state as user input. Show results in form of open and closed list.

Code:

```
from collections import deque

def bfs(graph, start, goal):
    open_list = deque([start])  # Queue for BFS
    closed_list = []  # List of explored nodes
    steps = []  # To track the state of open and closed lists at each step

    while open_list:
        current = open_list.popleft()  # Dequeue the first element
        closed_list.append(current)  # Add it to the closed list

        # Record the current state of open and closed lists
        steps.append((list(open_list), list(closed_list)))

        if current == goal:  # Goal state reached
            print("Goal reached!")
            return steps

        # Add neighbors to the open list (only if not already visited)
        for neighbor in graph[current]:
            if neighbor not in open_list and neighbor not in closed_list:
                open_list.append(neighbor)

    return steps
```

```
# Adjacency list for the given graph
graph = {
    'A': ['B', 'E', 'C'],
    'B': ['A', 'E'],
    'C': ['A', 'F', 'G'],
    'E': ['A', 'B', 'D'],
    'D': ['E'],
    'F': ['C'],
    'G': ['C']
}

# Get user input for the initial and final states
start = input("Enter the initial state: ").strip().upper()
goal = input("Enter the final state: ").strip().upper()

# Perform BFS
if start in graph and goal in graph:
    steps = bfs(graph, start, goal)

    # Print results
    print("\nSteps (Open List, Closed List):")
    for i, (open_list, closed_list) in enumerate(steps, start=1):
        print(f"Step {i}: Open List = {open_list}, Closed List = {closed_list}")
else:
    print("Invalid input! Make sure the initial and final states exist in the graph.")
```

Output:

```
Enter the initial state:  A
Enter the final state:  G
Goal reached!

Steps (Open List, Closed List):
Step 1: Open List = [], Closed List = ['A']
Step 2: Open List = ['E', 'C'], Closed List = ['A', 'B']
Step 3: Open List = ['C'], Closed List = ['A', 'B', 'E']
Step 4: Open List = ['D'], Closed List = ['A', 'B', 'E', 'C']
Step 5: Open List = ['F', 'G'], Closed List = ['A', 'B', 'E', 'C', 'D']
Step 6: Open List = ['G'], Closed List = ['A', 'B', 'E', 'C', 'D', 'F']
Step 7: Open List = [], Closed List = ['A', 'B', 'E', 'C', 'D', 'F', 'G']
```

3. Apply Depth First Search on the graph given in question 1. Considering the initial state is A and final state is G. Show results in form of open and closed list.

Code:

```python
def dfs(graph, start, goal):
    open_list = [start]
    closed_list = []
    steps = []

    while open_list:
        current = open_list.pop()
        if current not in closed_list:
            closed_list.append(current)


            steps.append((list(open_list), list(closed_list)))

            if current == goal:
                print("Goal reached!")
                return steps


            for neighbor in reversed(graph[current]):
                if neighbor not in closed_list:
                    open_list.append(neighbor)

    return steps
```

```python
graph = {
    'A': ['B', 'E', 'C'],
    'B': ['A', 'E'],
    'C': ['A', 'F', 'G'],
    'E': ['A', 'B', 'D'],
    'D': ['E'],
    'F': ['C'],
    'G': ['C']
}


start = 'A'
goal = 'G'
steps = dfs(graph, start, goal)
print("\nSteps (Open List, Closed List):")
for i, (open_list, closed_list) in enumerate(steps, start=1):
    print(f"Step {i}: Open List = {open_list}, Closed List = {closed_list}")
```

Output:

```
Goal reached!

Steps (Open List, Closed List):
Step 1: Open List = [], Closed List = ['A']
Step 2: Open List = ['C', 'E'], Closed List = ['A', 'B']
Step 3: Open List = ['C', 'E'], Closed List = ['A', 'B', 'E']
Step 4: Open List = ['C', 'E'], Closed List = ['A', 'B', 'E', 'D']
Step 5: Open List = [], Closed List = ['A', 'B', 'E', 'D', 'C']
Step 6: Open List = ['G'], Closed List = ['A', 'B', 'E', 'D', 'C', 'F']
Step 7: Open List = [], Closed List = ['A', 'B', 'E', 'D', 'C', 'F', 'G']
```

4. Using Question no. 1 apply DFS by taking initial and final state as user input. Show results in form of open and closed list.

Code:

```python
def dfs(graph, start, goal):
    open_list = [start]
    closed_list = []
    steps = []

    while open_list:
        current = open_list.pop()
        if current not in closed_list:
            closed_list.append(current)


            steps.append((list(open_list), list(closed_list)))

            if current == goal:
                print("Goal reached!")
                return steps


            for neighbor in reversed(graph[current]):
                if neighbor not in closed_list:
                    open_list.append(neighbor)

    return steps
```

```python
graph = {
    'A': ['B', 'E', 'C'],
    'B': ['A', 'E'],
    'C': ['A', 'F', 'G'],
    'E': ['A', 'B', 'D'],
    'D': ['E'],
    'F': ['C'],
    'G': ['C']
}


start = input("Enter the initial state: ").strip().upper()
goal = input("Enter the final state: ").strip().upper()

if start in graph and goal in graph:
    steps = dfs(graph, start, goal)


    print("\nSteps (Open List, Closed List):")
    for i, (open_list, closed_list) in enumerate(steps, start=1):
        print(f"Step {i}: Open List = {open_list}, Closed List = {closed_list}")
else:
    print("Invalid input! Make sure the initial and final states exist in the graph.")
```

Output:

```
Enter the initial state:  A
Enter the final state:  G
Goal reached!

Steps (Open List, Closed List):
Step 1: Open List = [], Closed List = ['A']
Step 2: Open List = ['C', 'E'], Closed List = ['A', 'B']
Step 3: Open List = ['C', 'E'], Closed List = ['A', 'B', 'E']
Step 4: Open List = ['C', 'E'], Closed List = ['A', 'B', 'E', 'D']
Step 5: Open List = [], Closed List = ['A', 'B', 'E', 'D', 'C']
Step 6: Open List = ['G'], Closed List = ['A', 'B', 'E', 'D', 'C', 'F']
Step 7: Open List = [], Closed List = ['A', 'B', 'E', 'D', 'C', 'F', 'G']
```