AI PRACTICAL NO. 4

Name:Girish Nhavkar

Roll no:9560

Div:TE COMPS A (BATCH B)

PART 1:

```
from collections import deque

def water_jug_bfs(capacity_jug1, capacity_jug2, target):
    visited_states = set()
    queue = deque([(0, 0, "Initial State")])  # Initial state: both jugs are
empty
    visited_states.add((0, 0))
    parent = {}  # Dictionary to keep track of the parent state for each state

    while queue:
        current_state = queue.popleft()
        jug1, jug2, action = current_state

        # Check if the goal state is reached
        if jug2 == target:
            print_steps(current_state, parent)
            return

        # Fill jug1
        fill_jug1 = (capacity_jug1, jug2, "Fill Jug1")
        if fill_jug1 not in visited_states:
            visited_states.add(fill_jug1)
            queue.append(fill_jug1)
            parent[fill_jug1] = current_state

        # Fill jug2
        fill_jug2 = (jug1, capacity_jug2, "Fill Jug2")
        if fill_jug2 not in visited_states:
            visited_states.add(fill_jug2)
            queue.append(fill_jug2)
            parent[fill_jug2] = current_state

        # Pour water from jug1 to jug2
        pour_jug1_to_jug2 = (max(0, jug1 - (capacity_jug2 - jug2)), min(jug2 +
jug1, capacity_jug2), "Pour Jug1 to Jug2")
        if pour_jug1_to_jug2 not in visited_states:
            visited_states.add(pour_jug1_to_jug2)
            queue.append(pour_jug1_to_jug2)
            parent[pour_jug1_to_jug2] = current_state
```

```
        # Pour water from jug2 to jug1
        pour_jug2_to_jug1 = (min(jug1 + jug2, capacity_jug1), max(0, jug2 -
(capacity_jug1 - jug1)), "Pour Jug2 to Jug1")
        if pour_jug2_to_jug1 not in visited_states:
            visited_states.add(pour_jug2_to_jug1)
            queue.append(pour_jug2_to_jug1)
            parent[pour_jug2_to_jug1] = current_state

        # Empty jug1
        empty_jug1 = (0, jug2, "Empty Jug1")
        if empty_jug1 not in visited_states:
            visited_states.add(empty_jug1)
            queue.append(empty_jug1)
            parent[empty_jug1] = current_state

        # Empty jug2
        empty_jug2 = (jug1, 0, "Empty Jug2")
        if empty_jug2 not in visited_states:
            visited_states.add(empty_jug2)
            queue.append(empty_jug2)
            parent[empty_jug2] = current_state

def print_steps(state, parent):
    steps = []
    while state[2] != "Initial State":
        steps.append(state)
        state = parent[state]
    steps.append((0, 0, "Initial State"))

    steps.reverse()
    for step in steps:
        print(f"{step[2]}: {step[0]} | {step[1]}")

# Example usage
capacity_jug1 = 3
capacity_jug2 = 4
target = 2

water_jug_bfs(capacity_jug1, capacity_jug2, target)



#output of code
```

```
PS C:\Girish\TE\AI> & "C:/Users/Girish Nhavkar/AppData/Local/Programs/Python/Python312/python.exe" c:/Girish/TE/AI/exp4/exp4_1.py
Initial State: 0 | 0
Fill Jug1: 3 | 0
Pour Jug1 to Jug2: 0 | 3
Fill Jug1: 3 | 3
Pour Jug1 to Jug2: 2 | 4
Empty Jug2: 2 | 0
Pour Jug1 to Jug2: 0 | 2
```

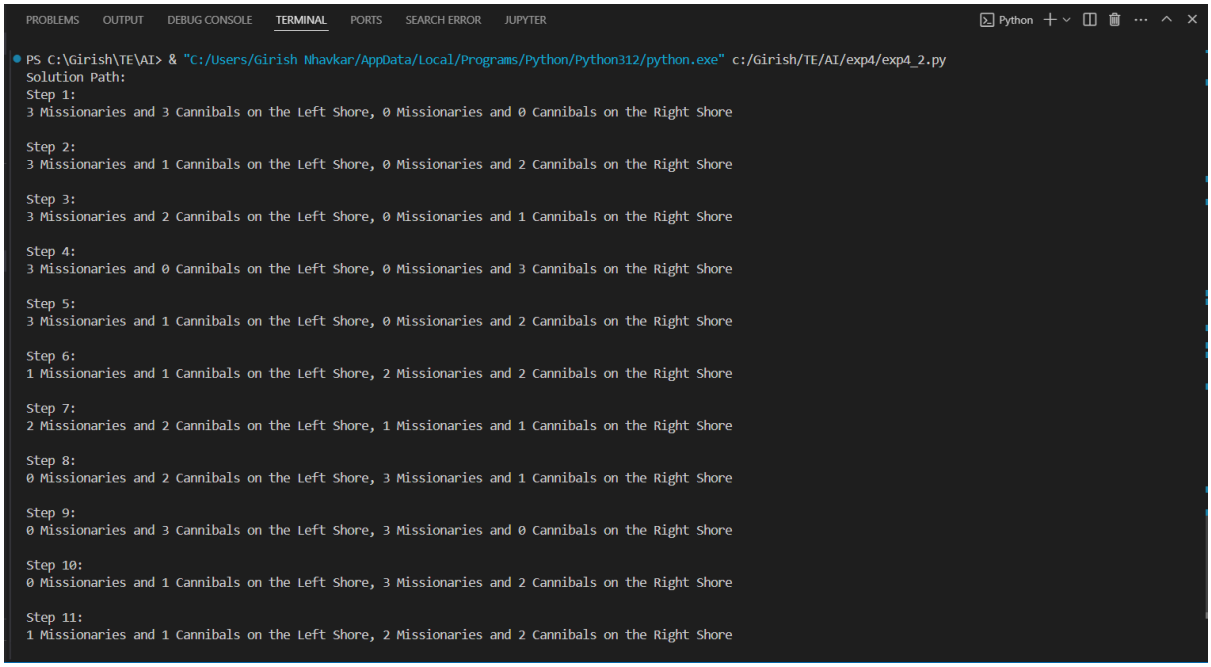
PART 2:

```python
from collections import deque

class State:
    def __init__(self, missionaries_left, cannibals_left, boat_left,
missionaries_right, cannibals_right):
        self.missionaries_left = missionaries_left
        self.cannibals_left = cannibals_left
        self.boat_left = boat_left
        self.missionaries_right = missionaries_right
        self.cannibals_right = cannibals_right

    def is_valid(self):
        if (
            0 <= self.missionaries_left <= 3
            and 0 <= self.cannibals_left <= 3
            and 0 <= self.missionaries_right <= 3
            and 0 <= self.cannibals_right <= 3
        ):
            if (
                self.missionaries_left >= self.cannibals_left
                or self.missionaries_left == 0
            ) and (
                self.missionaries_right >= self.cannibals_right
                or self.missionaries_right == 0
            ):
                return True
        return False

    def is_goal(self):
        return self.missionaries_left == 0 and self.cannibals_left == 0

    def __eq__(self, other):
        return (
            self.missionaries_left == other.missionaries_left
            and self.cannibals_left == other.cannibals_left
            and self.boat_left == other.boat_left
            and self.missionaries_right == other.missionaries_right
            and self.cannibals_right == other.cannibals_right
```

```
        )

    def __hash__(self):
        return hash((
            self.missionaries_left,
            self.cannibals_left,
            self.boat_left,
            self.missionaries_right,
            self.cannibals_right
        ))

def generate_next_states(current_state):
    next_states = []

    moves = [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)]

    for m, c in moves:
        if current_state.boat_left:
            new_state = State(
                current_state.missionaries_left - m,
                current_state.cannibals_left - c,
                1 - current_state.boat_left,
                current_state.missionaries_right + m,
                current_state.cannibals_right + c
            )
        else:
            new_state = State(
                current_state.missionaries_left + m,
                current_state.cannibals_left + c,
                1 - current_state.boat_left,
                current_state.missionaries_right - m,
                current_state.cannibals_right - c
            )

        if new_state.is_valid():
            next_states.append(new_state)

    return next_states

def bfs_search():
    start_state = State(3, 3, 1, 0, 0)
    goal_state = State(0, 0, 0, 3, 3)

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

    while queue:
        current_state, path = queue.popleft()
```

```
        if current_state.is_goal():
            return path

        if current_state not in visited:
            visited.add(current_state)

            next_states = generate_next_states(current_state)
            for next_state in next_states:
                if next_state not in visited:
                    queue.append((next_state, path + [current_state]))

    return None

def print_state_description(state):
    left_shore = f"{state.missionaries_left} Missionaries and
{state.cannibals_left} Cannibals on the Left Shore"
    right_shore = f"{state.missionaries_right} Missionaries and
{state.cannibals_right} Cannibals on the Right Shore"

    print(f"{left_shore}, {right_shore}\n")

if __name__ == "__main__":
    solution_path = bfs_search()

    if solution_path:
        print("Solution Path:")
        for i, state in enumerate(solution_path):
            print(f"Step {i + 1}:")
            print_state_description(state)

    else:
        print("No solution found.")
```

OP:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SEARCH ERROR
JUPYTER
Python
PS C:\Girish\TE\AI> & "C:/Users/Girish Nhavkar/AppData/Local/Programs/Python/Python312/python.exe" c:/Girish/TE/AI/exp4/exp4_2.py
Solution Path:
Step 1:
3 Missionaries and 3 Cannibals on the Left Shore, 0 Missionaries and 0 Cannibals on the Right Shore
Step 2:
3 Missionaries and 1 Cannibals on the Left Shore, 0 Missionaries and 2 Cannibals on the Right Shore
Step 3:
3 Missionaries and 2 Cannibals on the Left Shore, 0 Missionaries and 1 Cannibals on the Right Shore
Step 4:
3 Missionaries and 0 Cannibals on the Left Shore, 0 Missionaries and 3 Cannibals on the Right Shore
Step 5:
3 Missionaries and 1 Cannibals on the Left Shore, 0 Missionaries and 2 Cannibals on the Right Shore
Step 6:
1 Missionaries and 1 Cannibals on the Left Shore, 2 Missionaries and 2 Cannibals on the Right Shore
Step 7:
2 Missionaries and 2 Cannibals on the Left Shore, 1 Missionaries and 1 Cannibals on the Right Shore
Step 8:
0 Missionaries and 2 Cannibals on the Left Shore, 3 Missionaries and 1 Cannibals on the Right Shore
Step 9:
0 Missionaries and 3 Cannibals on the Left Shore, 3 Missionaries and 0 Cannibals on the Right Shore
Step 10:
0 Missionaries and 1 Cannibals on the Left Shore, 3 Missionaries and 2 Cannibals on the Right Shore
Step 11:
1 Missionaries and 1 Cannibals on the Left Shore, 2 Missionaries and 2 Cannibals on the Right Shore