Question:5

MISSIONARIES AND CANNIBALS PROBLEM

AIM

To solve the MISSIONARIES AND CANNIBALS problem using Python

ALGORITHM

1. The `is_valid` function checks if a given state is valid according to the problem constraints.

2. The `get_successors` function generates all possible valid successor states from a given state.

3. The `solve` function finds a solution from the starting state to the goal state using breadth-first search.

4. It initializes a queue (`q`) with the starting state and an empty path, and a set (`seen`) to keep track of visited states.

5. While the queue is not empty, it dequeues a state and its path.

6. If the dequeued state is the goal state, it returns the path.

7. Otherwise, it generates successor states, adds them to the queue if they haven't been seen before, and updates the set of seen states.

8. If no solution is found, it returns `None`.

9. The `print_solution` function prints the solution path if found, otherwise prints "No solution."

CODE

```
from collections import deque

def is_valid(s):
    # s: (missionaries left, cannibals left, boat, missionaries right, cannibals right)
    return all(0 <= x <= 3 for x in s[:5]) and (s[0] >= s[1] or s[0] == 0) and (s[3] >= s[4] or s[3] == 0)

def get_successors(s):
    transitions = [(-1, -1, -1, 1, 1), (-1, 0, -1, 1, 0), (0, -1, -1, 0, 1), (-2, 0, -1, 2, 0), (0, -2, -1, 0, 2)]
    if s[2] == 0:  # Adjust transitions for opposite boat direction
```

```
        transitions = [(x[3], x[4], 1, x[0], x[1]) for x in transitions]
    return [(s[0]+m, s[1]+c, (s[2]+b) % 2, s[3]-m, s[4]-c) for m, c, b, _, _ in transitions if
is_valid((s[0]+m, s[1]+c, (s[2]+b) % 2, s[3]-m, s[4]-c))]

def solve():
    start, goal = (3, 3, 1, 0, 0), (0, 0, 0, 3, 3)
    q = deque([(start, [])])
    seen = set([start])
    while q:
        state, path = q.popleft()
        if state == goal:
            return path + [goal]
        for next_state in get_successors(state):
            if next_state not in seen:
                seen.add(next_state)
                q.append((next_state, path + [state]))
    return None

def print_solution(solution):
    if not solution:
        print("No solution.")
    else:
        for s in solution:
            print(f"Left-> M{s[0]} C{s[1]}, Boat: {'left' if s[2] else 'right'}, Right-> M{s[3]}
C{s[4]}")

solution = solve()
print_solution(solution)
```

OUTPUT

```
Left-> M3 C3, Boat: left, Right-> M0 C0
Left-> M2 C2, Boat: right, Right-> M1 C1
Left-> M3 C2, Boat: left, Right-> M0 C1
Left-> M3 C0, Boat: right, Right-> M0 C3
Left-> M3 C1, Boat: left, Right-> M0 C2
Left-> M1 C1, Boat: right, Right-> M2 C2
Left-> M2 C2, Boat: left, Right-> M1 C1
Left-> M0 C2, Boat: right, Right-> M3 C1
Left-> M0 C3, Boat: left, Right-> M3 C0
Left-> M0 C1, Boat: right, Right-> M3 C2
Left-> M1 C1, Boat: left, Right-> M2 C2
Left-> M0 C0, Boat: right, Right-> M3 C3
```