PROGRAM-5

MISSIONARIES AND CANNIBALS PROBLEM

AIM:-

To write and execute the python program for the missionaries and cannibals program.

PROCEDURE:-

Define State Representation:

- Each state (missionaries_left, cannibals_left, boat, missionaries_right, cannibals_right) represents the configuration of missionaries and cannibals on both sides of the river and the position of the boat (0 for left and 1 for right).

Validity Check:

- The is_valid function checks if a state is valid, ensuring that there are no more cannibals than missionaries on either side of the river.

Generate Successors:

- The get_successors function generates successor states from the current state. It computes all possible valid transitions based on moving missionaries and cannibals across the river.

Breadth-First Search (BFS):

- The solve function performs BFS to find the solution. It explores the state space starting from the initial state and continues until it reaches the goal state.

Main Loop:

- Use a queue to perform BFS traversal.
- Pop states from the queue and expand them by generating successor states.
- Add valid successor states to the queue and continue until the goal state is reached.

Output Solution:

- Print the solution path if a solution is found.
- If no solution is found, print "No solution."

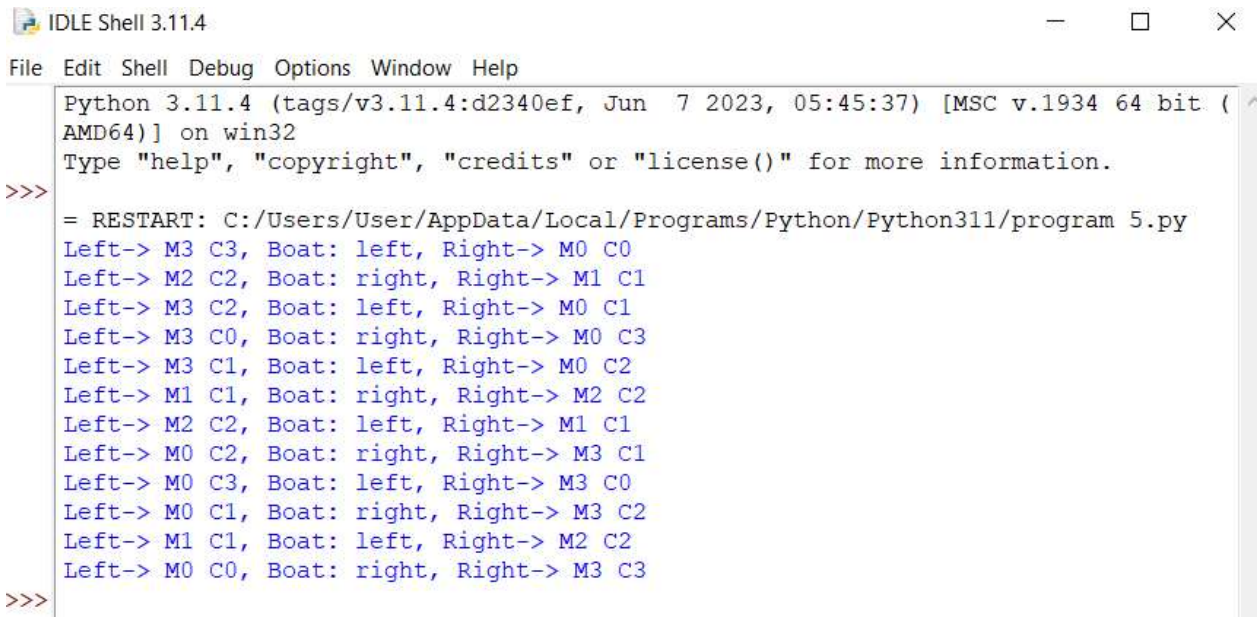
CODING:-

```
from collections import deque
```

```
def is_valid(s)

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:-



RESULT:-

Hence the program has been successfully executed and verified.