Program 5
Cannibal Missionary Problem

AIM :
To Create a python program to find a solution to the Cannibal Missionary Problem.

PROGRAM :

```
from collections import deque

class ComputeSolution:
    def __init__(self):
        pass

    def solve(self, initial_missionaries, initial_cannibals):
        class States:
            def __init__(self, left_missionaries, left_cannibals, right_missionaries,
right_cannibals, boat_position):
                self.left_missionaries = left_missionaries
                self.left_cannibals = left_cannibals
                self.right_missionaries = right_missionaries
                self.right_cannibals = right_cannibals
                self.boat_position = boat_position
                self.parent = None
            def __eq__(self, other):
                return (self.left_missionaries == other.left_missionaries and self.left_cannibals ==
other.left_cannibals and
                        self.right_missionaries == other.right_missionaries and self.right_cannibals
== other.right_cannibals and
                        self.boat_position == other.boat_position)
            def goal_state(self):
                if self.left_missionaries == 0 and self.left_cannibals == 0 and
self.right_missionaries == initial_missionaries and self.right_cannibals == initial_cannibals
and self.boat_position == "right":
                    return True
                else:
                    return False

            def valid_state(self):
                if (self.left_missionaries != 0 and self.left_cannibals > self.left_missionaries) or
(self.right_missionaries != 0 and self.right_cannibals > self.right_missionaries) or
self.left_missionaries < 0 or self.left_cannibals < 0 or self.right_missionaries < 0 or
self.right_cannibals < 0:
                    return False
                else:
                    return True
        def successors(curr_state):
```

```
        successor = []
        # Possible moves: Move 2 Missionaries, or 2 Cannibals, or 1 M + 1 C, or 1 M only, or
1 C only, to the other side
        possible_moves = [(2, 0), (0, 2), (1, 1), (1, 0), (0, 1)]
        if curr_state.boat_position == "left":  # boat moves from left to right
            for move in possible_moves:
                new_state = States(curr_state.left_missionaries - move[0],
curr_state.left_cannibals - move[1],
                                   curr_state.right_missionaries + move[0], curr_state.right_cannibals
+ move[1], "right")
                if new_state.valid_state():
                    successor.append(new_state)
                    new_state.parent = curr_state
        else:  # boat moves from right to left
            for move in possible_moves:
                new_state = States(curr_state.left_missionaries + move[0],
curr_state.left_cannibals + move[1],
                                   curr_state.right_missionaries - move[0], curr_state.right_cannibals -
move[1], "left")
                if new_state.valid_state():
                    successor.append(new_state)
                    new_state.parent = curr_state
        return successor

    def bfs():  # BFS
        initial_state = States(initial_missionaries, initial_cannibals, 0, 0, "left")  # starts at root
        if initial_state.goal_state():
            return initial_state
        queue = deque([])
        explored = []
        queue.append(initial_state)
        while queue:
            node = queue.popleft()
            if node.goal_state():
                return node
            explored.append(node)
            node_children = successors(node)
            for child in node_children:
                if (child not in explored) and (child not in queue):
                    queue.append(child)
        return None

    def find_moves(result):
        path = []
        final_path = []
        result_parent = result.parent
        while result_parent:
            move = (abs(result.left_missionaries - result_parent.left_missionaries),
```

```
                    abs(result.left_cannibals - result_parent.left_cannibals))
                path.append(move)
                result = result_parent
                result_parent = result.parent
            for i in range(len(path)):
                final_result = path[len(path) - 1 - i]
                final_path.append(final_result)
            return final_path

        solution = bfs()
        if solution:
            return find_moves(solution)
        else:
            return 'This iteration has no solution.'

def main():
    missionaries = input("Enter the number of missionaries: ")
    cannibals = input("Enter the number of cannibals: ")
    solution = ComputeSolution().solve(int(missionaries), int(cannibals))
    if type(solution) == str:
        print(solution)
    else:
        print('\nThese are the following steps of the solution: \n')
        iterator = 0
        for i in solution:
            if i[0] > 0 and i[1] > 0:
                print(f"Move {i[0]} missionaries and {i[1]} cannibals to the {'right' if ((iterator % 2) ==
0) else 'left'} side")
            elif i[0] > 0:
                print(f"Move {i[0]} missionaries to the {'right' if ((iterator % 2) == 0) else 'left'} side")
            elif i[1] > 0:
                print(f"Move {i[1]} cannibals to the {'right' if ((iterator % 2) == 0) else 'left'} side")
            iterator = iterator + 1
        print("\n--- End of solution ---")

if __name__ == "__main__":
        main()
```

OUTPUT:

```
Enter the number of missionaries: 3
Enter the number of cannibals: 3

These are the following steps of the solution:

Move 2 cannibals to the right side
Move 1 cannibals to the left side
Move 2 cannibals to the right side
Move 1 cannibals to the left side
Move 2 missionaries to the right side
Move 1 missionaries and 1 cannibals to the left side
Move 2 missionaries to the right side
Move 1 cannibals to the left side
Move 2 cannibals to the right side
Move 1 missionaries to the left side
Move 1 missionaries and 1 cannibals to the right side
```

RESULT:

The Program has successfully been executed.