TITLE: Write the python program for Missionaries Cannibal problem

CODE:

```
from collections import deque


class State:
    def __init__(self, missionaries, cannibals, boat, parent=None):
        self.missionaries = missionaries
        self.cannibals = cannibals
        self.boat = boat
        self.parent = parent

    def is_valid(self):
        if self.missionaries < 0 or self.missionaries > 3:
            return False
        if self.cannibals < 0 or self.cannibals > 3:
            return False
        if self.missionaries > 0 and self.missionaries < self.cannibals:
            return False
        if self.missionaries < 3 and (3 - self.missionaries) < (3 - self.cannibals):
            return False
```

```
        return True

    def is_goal(self):
        return self.missionaries == 0 and self.cannibals == 0 and
self.boat == 0

    def __eq__(self, other):
        return self.missionaries == other.missionaries and
self.cannibals == other.cannibals and self.boat == other.boat

    def __hash__(self):
        return hash((self.missionaries, self.cannibals, self.boat))

    def __str__(self):
        return f'M: {self.missionaries}, C: {self.cannibals}, B:
{self.boat}'

def successors(state):
    children = []
    if state.boat == 1:
        # Boat is on the initial bank
        for m in range(3):
            for c in range(3):
```

```
            if 1 <= m + c <= 2:
                new_state = State(state.missionaries - m,
state.cannibals - c, 0, state)
                if new_state.is_valid():
                    children.append(new_state)
    else:
        # Boat is on the other bank
        for m in range(3):
            for c in range(3):
                if 1 <= m + c <= 2:
                    new_state = State(state.missionaries + m,
state.cannibals + c, 1, state)
                    if new_state.is_valid():
                        children.append(new_state)
    return children


def bfs():
    start_state = State(3, 3, 1)
    visited = set()
    queue = deque([start_state])


    while queue:
```

```
        state = queue.popleft()
        if state.is_goal():
            return state
        visited.add(state)
        for child in successors(state):
            if child not in visited:
                queue.append(child)
    return None

def print_solution(solution):
    path = []
    while solution:
        path.append(solution)
        solution = solution.parent
    for t in path[::-1]:
        print(t)

def main():
    solution = bfs()
    if solution:
        print("Solution found:")
        print_solution(solution)
```

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


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

OUTPUT:

```
>>>
========================= RESTART: D:\python\mission.py ========================
Solution found:
M: 3, C: 3, B: 1
M: 3, C: 1, B: 0
M: 3, C: 2, B: 1
M: 3, C: 0, B: 0
M: 3, C: 1, B: 1
M: 1, C: 1, B: 0
M: 2, C: 2, B: 1
M: 0, C: 2, B: 0
M: 0, C: 3, B: 1
M: 0, C: 1, B: 0
M: 0, C: 2, B: 1
M: 0, C: 0, B: 0
>>>
```