**EXPERIMENT 5:**

from collections import deque

class State:

def \_\_init\_\_(self, missionaries\_left, cannibals\_left, boat\_left, path=[]):

self.m\_left = missionaries\_left

self.c\_left = cannibals\_left

self.boat = boat\_left # 1 if boat is on left, 0 if on right

self.path = path

def is\_valid(self):

m\_right = 3 - self.m\_left

c\_right = 3 - self.c\_left

if self.m\_left < 0 or self.c\_left < 0 or m\_right < 0 or c\_right < 0:

return False

if (self.m\_left > 0 and self.m\_left < self.c\_left) or (m\_right > 0 and m\_right < c\_right):

return False

return True

def is\_goal(self):

return self.m\_left == 0 and self.c\_left == 0 and self.boat == 0

def get\_successors(self):

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

successors = []

for m, c in moves:

if self.boat == 1:

new\_state = State(self.m\_left - m, self.c\_left - c, 0, self.path + [self])

else:

new\_state = State(self.m\_left + m, self.c\_left + c, 1, self.path + [self])

if new\_state.is\_valid():

successors.append(new\_state)

return successors

def \_\_str\_\_(self):

return f"Left: M={self.m\_left}, C={self.c\_left} | Right: M={3 - self.m\_left}, C={3 - self.c\_left} | Boat: {'Left' if self.boat else 'Right'}"

def bfs():

start = State(3, 3, 1)

queue = deque([start])

visited = set()

while queue:

current = queue.popleft()

if current.is\_goal():

return current.path + [current]

visited.add((current.m\_left, current.c\_left, current.boat))

for successor in current.get\_successors():

if (successor.m\_left, successor.c\_left, successor.boat) not in visited:

queue.append(successor)

return None

# Run the search

solution = bfs()

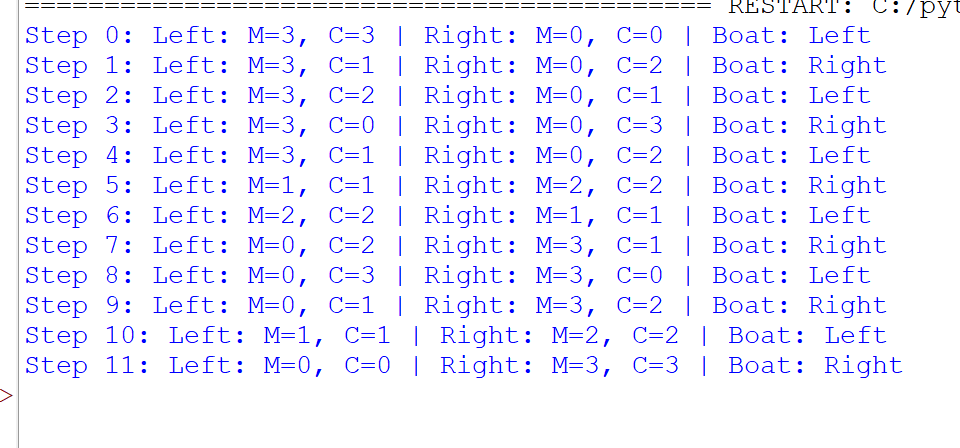
if solution:

for step, state in enumerate(solution):

print(f"Step {step}: {state}")

else:

print("No solution found.")

**OUTPUT:**