**Experiment 5:**

**Aim:** To write a python program to implement a Missionaries and Cannibal program.

**Algorithm:**

1. Represent the state as (m, c, b) with the goal state (0, 0, 0).
2. Ensure missionaries are not outnumbered by cannibals on either side, and boat capacity is not exceeded.
3. Generate valid moves: (1, 0), (0, 1), (2, 0), (0, 2), (1, 1).
4. Use BFS to explore states from (3, 3, 1), avoiding revisiting states.
5. Stop at (0, 0, 0), trace the solution path, and display the sequence of moves.

**Program:**

from collections import deque

# Check if the state is valid

def is\_valid(state, total\_missionaries, total\_cannibals):

m\_left, c\_left, m\_right, c\_right = state

# Condition 1: The number of missionaries and cannibals must be between 0 and the total number of missionaries and cannibals

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

return False

# Condition 2: Missionaries should never be outnumbered by cannibals on either side

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

return False

# Condition 3: Missionaries and cannibals should never exceed the total number

if m\_left + m\_right != total\_missionaries or c\_left + c\_right != total\_cannibals:

return False

return True

# BFS to solve the problem

def missionaries\_and\_cannibals(total\_missionaries, total\_cannibals):

initial\_state = (total\_missionaries, total\_cannibals, 0, 0) # (m\_left, c\_left, m\_right, c\_right)

goal\_state = (0, 0, total\_missionaries, total\_cannibals)

queue = deque([(initial\_state, [])]) # Queue stores tuples of (state, path)

visited = set() # To avoid revisiting states

visited.add(initial\_state)

while queue:

current\_state, path = queue.popleft()

# If we reached the goal state, return the solution path

if current\_state == goal\_state:

print("Solution found!")

for step in path:

print(step)

return

# Generate possible next states

m\_left, c\_left, m\_right, c\_right = current\_state

# Possible moves: (number of missionaries, number of cannibals)

moves = [

(1, 0), # 1 missionary crosses

(0, 1), # 1 cannibal crosses

(2, 0), # 2 missionaries cross

(0, 2), # 2 cannibals cross

(1, 1) # 1 missionary and 1 cannibal cross

]

for m\_move, c\_move in moves:

# Calculate new states based on the boat moving

if m\_left >= m\_move and c\_left >= c\_move:

new\_state = (m\_left - m\_move, c\_left - c\_move, m\_right + m\_move, c\_right + c\_move)

if is\_valid(new\_state, total\_missionaries, total\_cannibals) and new\_state not in visited:

visited.add(new\_state)

new\_path = path + [f"Move {m\_move} missionaries and {c\_move} cannibals."]

queue.append((new\_state, new\_path))

print("No solution found.")

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

total\_missionaries = int(input("Enter the number of missionaries: "))

total\_cannibals = int(input("Enter the number of cannibals: "))

# Ensure valid input

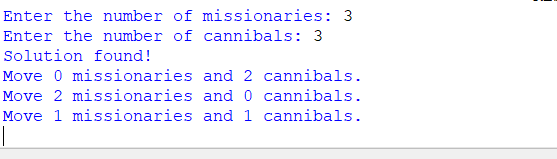
if total\_missionaries < 1 or total\_cannibals < 1:

print("The number of missionaries and cannibals must be greater than or equal to 1.")

else:

missionaries\_and\_cannibals(total\_missionaries, total\_cannibals)

**Output:**

****

**Result:** Thus, the program was successfully completed using python programming.