**EXPERIMENT 5:**

Write the python program for Missionaries Cannibal problem.

**AIM:**

The aim is to solve the Missionaries Cannibal problem using python program.

**PROGRAM:**

from queue import Queue

class State:

def \_\_init\_\_(self, missionaries, cannibals, boat):

self.missionaries = missionaries

self.cannibals = cannibals

self.boat = boat

def is\_valid(self):

if self.missionaries < 0 or self.cannibals < 0 or self.missionaries > 3 or self.cannibals > 3:

return False

if self.cannibals > self.missionaries > 0 or (3 - self.cannibals) > (3 - self.missionaries) > 0:

return False

return True

def is\_goal(self):

return self.missionaries == 0 and self.cannibals == 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 get\_neighbors(current\_state):

neighbors = []

possible\_moves = [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)]

for move in possible\_moves:

new\_state = State(

current\_state.missionaries - move[0] \* current\_state.boat,

current\_state.cannibals - move[1] \* current\_state.boat,

1 - current\_state.boat

)

new\_state\_on\_other\_side = State(

current\_state.missionaries + move[0] \* (1 - current\_state.boat),

current\_state.cannibals + move[1] \* (1 - current\_state.boat),

1 - current\_state.boat

)

if new\_state.is\_valid() and new\_state\_on\_other\_side.is\_valid():

neighbors.append(new\_state)

return neighbors

def breadth\_first\_search():

initial\_state = State(3, 3, 1)

goal\_state = State(0, 0, 0)

frontier = Queue()

frontier.put(initial\_state)

came\_from = {}

came\_from[initial\_state] = None

while not frontier.empty():

current\_state = frontier.get()

if current\_state.is\_goal():

path = []

while current\_state:

path.append(current\_state)

current\_state = came\_from[current\_state]

path.reverse()

return path

for next\_state in get\_neighbors(current\_state):

if next\_state not in came\_from:

frontier.put(next\_state)

came\_from[next\_state] = current\_state

return None

def print\_solution(path):

for t, state in enumerate(path):

print(f"Step {t + 1}:")

print(f"On the side {state.boat} - Missionaries: {state.missionaries}, Cannibals: {state.cannibals}")

print(f"On the other side {1 - state.boat} - Missionaries: {3 - state.missionaries}, Cannibals: {3 - state.cannibals}")

print()

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

solution\_path = breadth\_first\_search()

if solution\_path:

print("Solution found!")

print\_solution(solution\_path)

else:

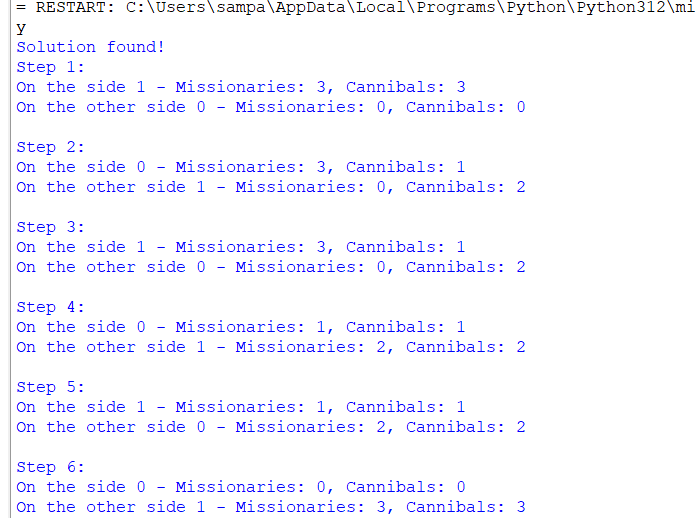
print("No solution found.")

**INPUT:**

No. of. Missionaries:3

No. of. Cannibals:3

**OUTPUT:**

****

**RESULT:**

Missionaries and cannibals program executed successfully.