EXPERIMENT:5

Missionaries Cannibal problem

AIM: To write the python program for Missionaries Cannibal problem

**Algorithm:**

1. Define initial and goal states: **start** represents the initial state where three elements are on one bank of the river, and **end** represents the goal state where all elements are on the opposite bank.
2. Define the **do\_action** function to perform an action (crossing the river) from one state to another.
3. Implement helper functions **is\_legal** to check if a state is within the problem's constraints, **is\_bank\_safe** to check if the bank configuration is safe, and **is\_state\_safe** to check if both banks are safe in a particular state.
4. Define **next\_possible\_actions** to generate possible actions that can be taken from a given state.
5. Implement the **solve** function using depth-first search (DFS) to explore possible paths from the initial state to the goal state recursively.
6. Print out all solutions found.

CODE:

start,end =[3,3,1],[0,0,0]

def do\_action(state,action):

if state[2] == 1:

return [state[i] - action[i] for i in range(3)]

else:

return [state[i] + action[i] for i in range(3)]

def is\_legal(state):

if 0 <= state[0] <= 3 and 0 <= state[1] <= 3:

return True

else:

return False

def is\_bank\_safe(bank):

if bank[1] > bank[0] and bank[0] != 0:

return False

else:

return True

def is\_state\_safe(state):

other\_bank = [start[i]-state[i] for i in range(3)]

if is\_bank\_safe(state) and is\_bank\_safe(other\_bank) :

return True

else:

return False

def next\_possible\_actions(state):

actions = [[1,0,1],[0,1,1],[1,1,1],[2,0,1],[0,2,1]]

moves = []

for i in actions:

j = do\_action(state,i)

if is\_legal(j) and is\_state\_safe(j):

moves.append(j)

return moves

solutions = []

def solve(next\_action,path):

\_path = path.copy()

if next\_action == end:

\_path.append(next\_action)

solutions.append(\_path)

return

elif next\_action in path:

return

else:

\_path.append(next\_action)

for i in next\_possible\_actions(next\_action):

solve(i,\_path)

solve([3,3,1],[])

print(\*solutions,sep="\n")

OUTPUT:

[[3, 3, 1], [2, 2, 0], [3, 2, 1], [3, 0, 0], [3, 1, 1], [1, 1, 0], [2, 2, 1], [0, 2, 0], [0, 3, 1], [0, 1, 0], [1, 1, 1], [0, 0, 0]]

[[3, 3, 1], [2, 2, 0], [3, 2, 1], [3, 0, 0], [3, 1, 1], [1, 1, 0], [2, 2, 1], [0, 2, 0], [0, 3, 1], [0, 1, 0], [0, 2, 1], [0, 0, 0]]

[[3, 3, 1], [3, 1, 0], [3, 2, 1], [3, 0, 0], [3, 1, 1], [1, 1, 0], [2, 2, 1], [0, 2, 0], [0, 3, 1], [0, 1, 0], [1, 1, 1], [0, 0, 0]]

[[3, 3, 1], [3, 1, 0], [3, 2, 1], [3, 0, 0], [3, 1, 1], [1, 1, 0], [2, 2, 1], [0, 2, 0], [0, 3, 1], [0, 1, 0], [0, 2, 1], [0, 0, 0]]