**AIM: Write a program to implement to implement Missionary Cannibal Problem.**

**CODE:**

import math

class State():

def \_\_init\_\_(self, cannibalLeft, missionaryLeft, boat, cannibalRight, missionaryRight):

self.cannibalLeft = cannibalLeft

self.missionaryLeft = missionaryLeft

self.boat = boat

self.cannibalRight = cannibalRight

self.missionaryRight = missionaryRight

self.parent = None

def is\_goal(self):

if self.cannibalLeft == 0 and self.missionaryLeft == 0:

return True

else:

return False

def is\_valid(self):

if self.missionaryLeft >= 0 and self.missionaryRight >= 0 \

and self.cannibalLeft >= 0 and self.cannibalRight >= 0 \

and (self.missionaryLeft == 0 or self.missionaryLeft >= self.cannibalLeft) \

and (self.missionaryRight == 0 or self.missionaryRight >= self.cannibalRight):

return True

else:

return False

def \_\_eq\_\_(self, other):

return self.cannibalLeft == other.cannibalLeft and self.missionaryLeft == other.missionaryLeft \

and self.boat == other.boat and self.cannibalRight == other.cannibalRight \

and self.missionaryRight == other.missionaryRight

def \_\_hash\_\_(self):

return hash((self.cannibalLeft, self.missionaryLeft, self.boat, self.cannibalRight, self.missionaryRight))

def successors(cur\_state):

children = [];

if cur\_state.boat == 'left':

new\_state = State(cur\_state.cannibalLeft, cur\_state.missionaryLeft - 2, 'right',

cur\_state.cannibalRight, cur\_state.missionaryRight + 2)

## Two missionaries cross left to right.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

new\_state = State(cur\_state.cannibalLeft - 2, cur\_state.missionaryLeft, 'right',

cur\_state.cannibalRight + 2, cur\_state.missionaryRight)

## Two cannibals cross left to right.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

new\_state = State(cur\_state.cannibalLeft - 1, cur\_state.missionaryLeft - 1, 'right',

cur\_state.cannibalRight + 1, cur\_state.missionaryRight + 1)

## One missionary and one cannibal cross left to right.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

new\_state = State(cur\_state.cannibalLeft, cur\_state.missionaryLeft - 1, 'right',

cur\_state.cannibalRight, cur\_state.missionaryRight + 1)

## One missionary crosses left to right.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

new\_state = State(cur\_state.cannibalLeft - 1, cur\_state.missionaryLeft, 'right',

cur\_state.cannibalRight + 1, cur\_state.missionaryRight)

## One cannibal crosses left to right.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

else:

new\_state = State(cur\_state.cannibalLeft, cur\_state.missionaryLeft + 2, 'left',

cur\_state.cannibalRight, cur\_state.missionaryRight - 2)

## Two missionaries cross right to left.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

new\_state = State(cur\_state.cannibalLeft + 2, cur\_state.missionaryLeft, 'left',

cur\_state.cannibalRight - 2, cur\_state.missionaryRight)

## Two cannibals cross right to left.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

new\_state = State(cur\_state.cannibalLeft + 1, cur\_state.missionaryLeft + 1, 'left',

cur\_state.cannibalRight - 1, cur\_state.missionaryRight - 1)

## One missionary and one cannibal cross right to left.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

new\_state = State(cur\_state.cannibalLeft, cur\_state.missionaryLeft + 1, 'left',

cur\_state.cannibalRight, cur\_state.missionaryRight - 1)

## One missionary crosses right to left.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

new\_state = State(cur\_state.cannibalLeft + 1, cur\_state.missionaryLeft, 'left',

cur\_state.cannibalRight - 1, cur\_state.missionaryRight)

## One cannibal crosses right to left.

if new\_state.is\_valid():

new\_state.parent = cur\_state

children.append(new\_state)

return children

def breadth\_first\_search():

initial\_state = State(3,3,'left',0,0)

if initial\_state.is\_goal():

return initial\_state

frontier = list()

explored = set()

frontier.append(initial\_state)

while frontier:

state = frontier.pop(0)

if state.is\_goal():

return state

explored.add(state)

children = successors(state)

for child in children:

if (child not in explored) or (child not in frontier):

frontier.append(child)

return None

def print\_solution(solution):

path = []

path.append(solution)

parent = solution.parent

while parent:

path.append(parent)

parent = parent.parent

for t in range(len(path)):

state = path[len(path) - t - 1]

print("(" + str(state.cannibalLeft) + "," + str(state.missionaryLeft) \

+ "," + state.boat + "," + str(state.cannibalRight) + "," + \

str(state.missionaryRight) + ")")

def main():

solution = breadth\_first\_search()

print("Missionaries and Cannibals solution:")

print("(cannibalLeft,missionaryLeft,boat,cannibalRight,missionaryRight)")

print\_solution(solution)

# if called from the command line, call main()

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

main()

**OUTPUT:**

