NAME: SHREEJA BISWAS

DEPARTMENT: CSE YEAR: 4th

ROLL NUMBER: 13

SEMESTER: 7th

SUBJECT: AI ASSIGNMENT

• QUESTION:

In this problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, that the missionaries present on the bank cannot be outnumbered by cannibals. The boat cannot cross the river by itself with no people on board. Implement the above problem of AI using python/MATLAB/C.

• 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) 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) 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) 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) 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) 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) 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) 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) 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)

            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) 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 __name__ == "__main__":
main()
```