**EXPERIMENT NO : 03 DATE : 20/02/24**

**Aim**: Program to implement Missionaries and Cannibals problem using BFS and DFS algorithms.

**Theory**:

Missionaries and Cannibals is a problem in which ‘n’ missionaries and ‘n’ cannibals want to cross from

the left bank of a river to the right bank of the river. There is a boat on the left bank, but it only carries

at most two people at a time (and can never cross with zero people). If cannibals ever outnumber

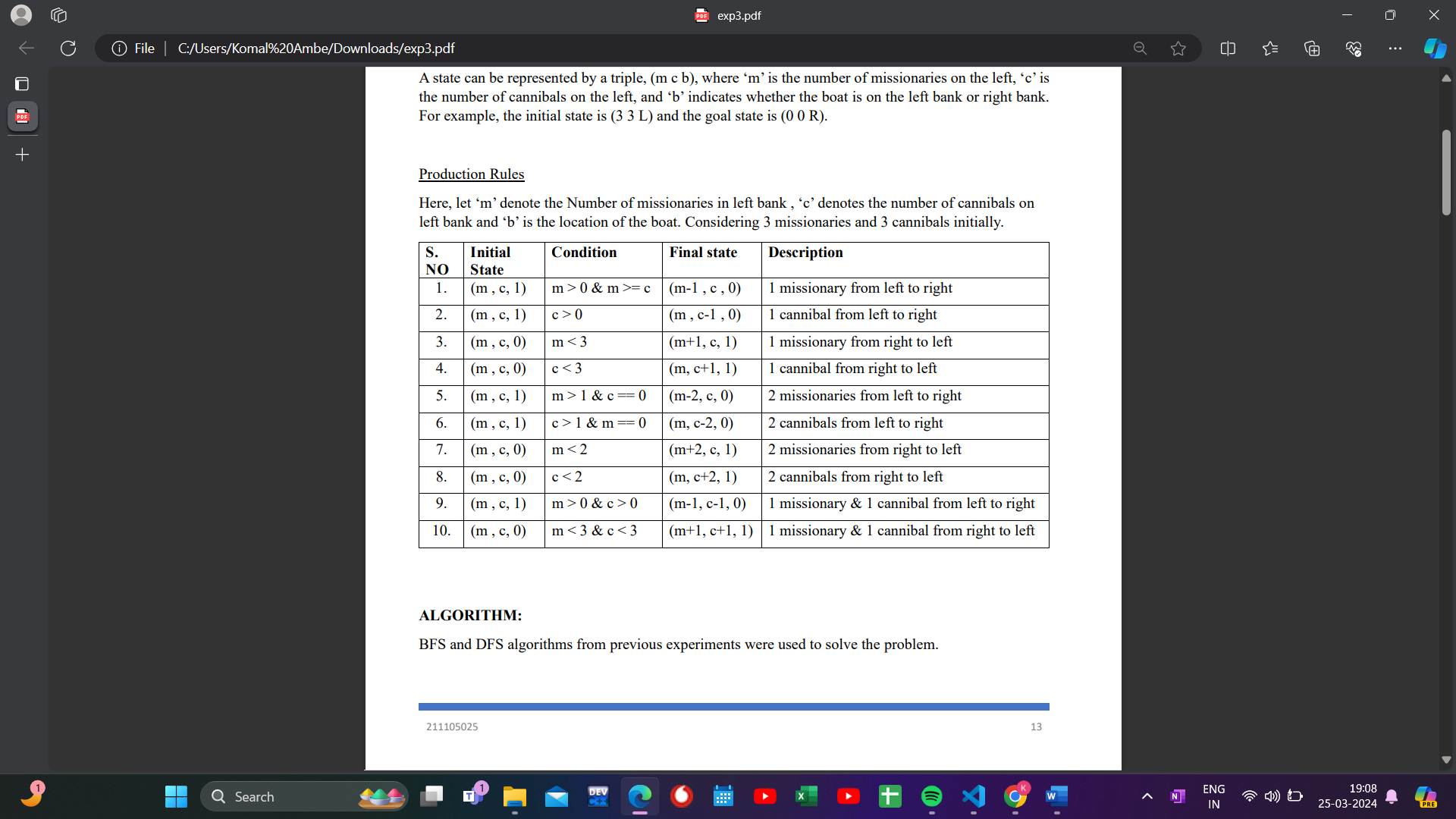
missionaries on either bank, the cannibals will eat the missionaries

A state can be represented by a triple, (m, c, b), where ‘m’ is the number of missionaries on the left, ‘c’ is

the number of cannibals on the left, and ‘b’ indicates whether the boat is on the left bank or right bank.

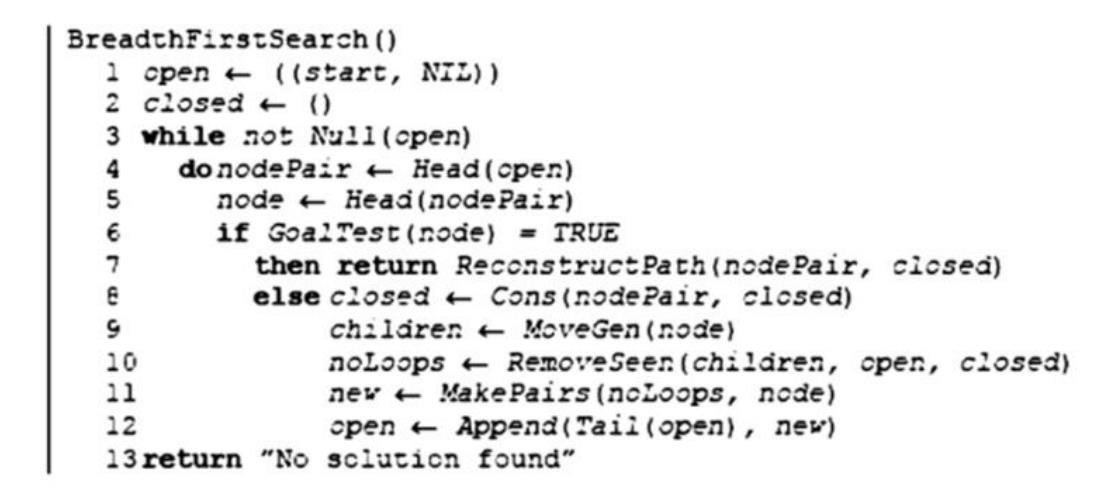
For example, the initial state is (3, 3, L) and the goal state is (0, 0, R).

**Production Rules:**



1. **Solution using BFS**

**Algorithm:**



**Code:**

from collections import deque

visited = set()

def check(x, y, n):

if x < 0 or y < 0 or x > n or y > n:

return False

if (x >= y or x == 0 or y == 0) and (

(n - x) >= (n - y) or (n - x) == 0 or (n - y) == 0

):

return True

else:

return False

def misscan(m, c, b, n):

global f

queue = deque([(m, c, b, [])])

solutions = []

while queue:

m, c, b, path = queue.popleft() # dequeue a state

if (m, c, b) in visited:

continue

path.append((m, c, b))

# end state

if (m, c, b) == (0, 0, "R"):

solutions.append(path)

continue # we continue from here because if we don’t then we will add the final state in vis and never be able to go to it again

visited.add((m, c, b))

if b == "L":

# send 1 miss

if check(m - 1, c, n):

queue.append((m - 1, c, "R", path[:]))

# send 2 miss

if check(m - 2, c, n):

queue.append((m - 2, c, "R", path[:]))

# send 1 cann

if check(m, c - 1, n):

queue.append((m, c - 1, "R", path[:]))

# send 2 cann

if check(m, c - 2, n):

queue.append((m, c - 2, "R", path[:]))

# send 1 miss 1 cann

if check(m - 1, c - 1, n):

queue.append((m - 1, c - 1, "R", path[:]))

else:

# send 1 miss

if check(m + 1, c, n):

queue.append((m + 1, c, "L", path[:]))

# send 2 miss

if check(m + 2, c, n):

queue.append((m + 2, c, "L", path[:]))

# send 1 cann

if check(m, c + 1, n):

queue.append((m, c + 1, "L", path[:]))

# send 2 cann

if check(m, c + 2, n):

queue.append((m, c + 2, "L", path[:]))

# send 1 miss 1 cann

if check(m + 1, c + 1, n):

queue.append((m + 1, c + 1, "L", path[:]))

return solutions

def input\_validity(m, c):

if c > m > 0 or m == 0:

return False

return True

while True:

num\_miss = int(input("Enter number of missionaries: "))

num\_cann = int(input("Enter number of cannibals: "))

boat\_capacity = int(input("Enter the boat capacity: "))

if not input\_validity(num\_miss, num\_cann):

print("Invalid input: Number of cannibals cannot exceed the number of missionaries.")

continue

if num\_miss < boat\_capacity and num\_cann < boat\_capacity:

print("No solution possible. Both missionaries and cannibals are less than boat capacity.")

continue

solutions = misscan(num\_miss, num\_cann, "L", max(num\_miss, num\_cann))

if not solutions:

print("No solution could be found.")

else:

print("Solutions:")

for solution in solutions:

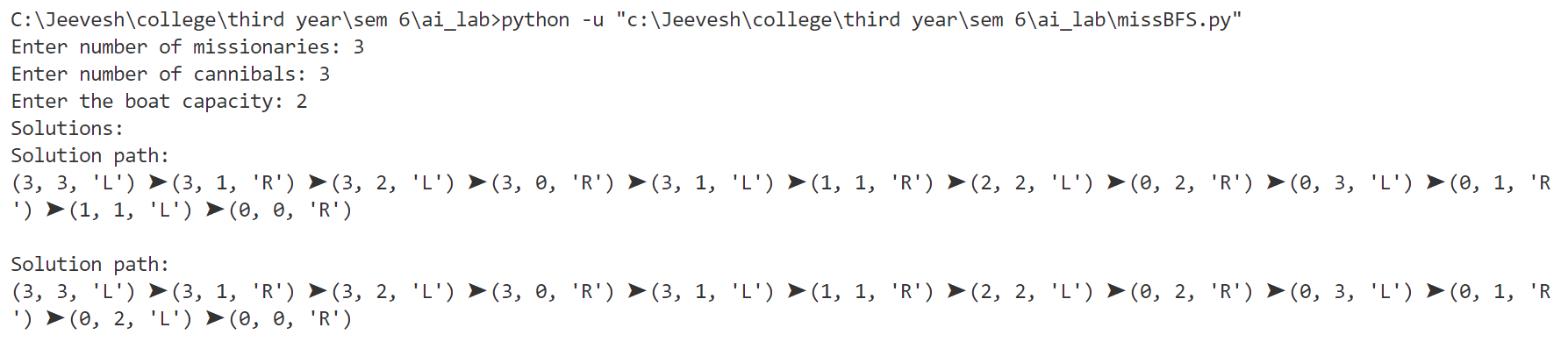
print("Solution path:")

print(\*solution, sep=" ➤ ")

print()

break

**Output:**



1. **Solution using DFS**

**Algorithm:**



**Code:**

f = 0

visited = []

def check(x, y, n):

if x < 0 or y < 0 or x > n or y > n:

return False

if (x >= y or x == 0 or y == 0) and (

(n - x) >= (n - y) or (n - x) == 0 or (n - y) == 0

):

return True

else:

return False

def misscan(i, j, boat\_capacity, bank, path, n):

global f

if (i, j, bank) in visited:

return

path.append((i, j, bank))

# end state

if (i, j, bank) == (0, 0, "R"):

f = 1

print("Solution:")

print(\*path, sep=" ➤ ")

print()

path.pop()

return

visited.append((i, j, bank))

if bank == "L":

# send missionaries and cannibals in all possible combinations

for miss in range(boat\_capacity + 1):

for cann in range(boat\_capacity - miss + 1):

if miss + cann > 0 and miss + cann <= boat\_capacity:

if check(i - miss, j - cann, n):

misscan(i - miss, j - cann, boat\_capacity, "R", path, n)

else:

# send missionaries and cannibals in all possible combinations

for miss in range(boat\_capacity + 1):

for cann in range(boat\_capacity - miss + 1):

if miss + cann > 0 and miss + cann <= boat\_capacity:

if check(i + miss, j + cann, n):

misscan(i + miss, j + cann, boat\_capacity, "L", path, n)

path.pop()

def validate\_input(missionaries, cannibals, boat\_capacity):

if cannibals > missionaries > 0 or missionaries <= 0 or cannibals <= 0 or boat\_capacity <= 0:

print("Invalid input: Number of cannibals cannot exceed missionaries, and all inputs must be positive.")

return False

return True

def get\_input():

missionaries = int(input("Enter number of missionaries: "))

cannibals = int(input("Enter number of cannibals: "))

boat\_capacity = int(input("Enter boat capacity: "))

return missionaries, cannibals, boat\_capacity

missionaries, cannibals, boat\_capacity = get\_input()

if not validate\_input(missionaries, cannibals, boat\_capacity):

exit()

p = []

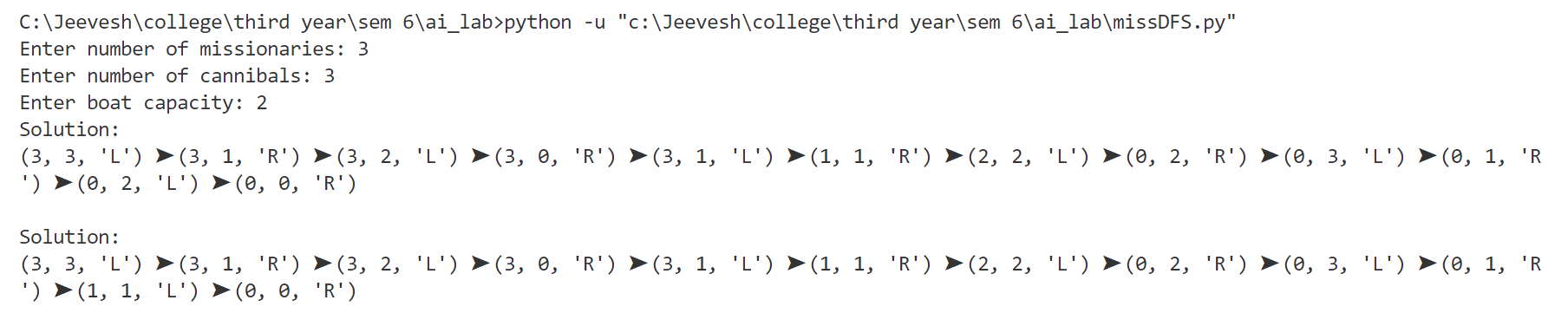
n = max(missionaries, cannibals)

misscan(missionaries, cannibals, boat\_capacity, "L", p, n)

if f == 0:

print("No solution could be found.")

**Output:**

****

**Conclusion :** Missionaries and cannibals using Breadth First Search and Depth First Search was implemented.