Experiment No : 3 Date:

Missionary and Cannibal Problem

Aim : To solve Missionary and Cannibal Problem using BFS and DFS algorithms.

THEORY:

Problem Statement:

The Missionary and Cannibal problem is a state space search puzzle that involves transporting three missionaries and three cannibals across a river using a boat with a capacity of two. At any point during the crossing, there cannot be more cannibals than missionaries on either bank or in the boat. The objective is to devise a sequence of boat trips that safely transports all individuals from one side of the river to the other while adhering to these constraints, ensuring the safety of the missionaries throughout the process. This problem challenges individuals to explore and navigate through different states of the system to find a solution.

State Space:

The state space for the Missionary and Cannibal problem consists of all possible configurations of

missionaries and cannibals on both sides of the river, as well as the position of the boat. Each state is

represented by the number of missionaries and cannibals on the left bank and whether the boat is on the left bank or the right bank. The state space is constrained by the number of missionaries, cannibals, and the

boat's capacity.

The state space for this problem can be described as the set of tuples (M, C, boat), where:

* M represents the number of missionaries on the left bank,
* C represents the number of cannibals on the left bank,
* boat represents the position of the boat (either 0 or 1 where 0->left and 1->right).

The initial state is (M, C, 0) and the goal state is (0, 0, 1), indicating all missionaries and cannibals have safely crossed to the right bank.

Production Rules:

1. Going from Left to Right such that x>=y and M-x>=C-y A] 2 Cannibals (x,y,0)->(x,y-2,1)
   1. 2 Missionary (x,y,0)->(x-2,y,1)
   2. 1 Cannibal and 1 missionary (x,y,0)->(x-1,y-1,1) D] 1 Missionary (x,y,0)->(x-1,y,1)

E] 1 Cannibal (x,y,0)->(x,y-1,1)

1. Going from Right to Left such that x>=y and M-x>=C-y A] 2 Cannibals (x,y,1)->(x,y+2,0)
2. 2 Missionary (x,y,1)->(x+2,y,0)
3. 1 Cannibal and 1 missionary (x,y,1)->(x+1,y+1,0) D] 1 Missionary (x,y,1)->(x+1,y,0)

E] 1 Cannibal (x,y,1)->(x,y+1,0)

# ALGORITHMS:

1. BFS-ALGORITHM BFS()

Open=((start , Nil)) , closed= () While Open is not empty do Nodepair=head(open)

node = head(Nodepair)

if Goalstate(node) == True then

Return reconstructpath(Nodepair,closed)

else

closed=cons(Nodepair,closed) children=movegen(node)

noloop= removeseen(children,open,closed) new=makepair(noloop,node)

Open=Append(tail(Open),new)

Return Failure

1. DFS-ALGORITHM

DFS()

Open=((start , Nil)) , closed= () While Open is not empty do Nodepair=head(open)

node = head(Nodepair)

if Goalstate(node) == True then

Return reconstructpath(Nodepair,closed) else

closed=cons(Nodepair,closed) children=movegen(node)

noloop= removeseen(children,open,closed) new=makepair(noloop,node)

Open=Append(new,tail(Open)) Return Failure

CODE:

1]Missionary and Cannibals Using BFS

from queue import deque

m = int(input("No. of Missionaires : "))

c = int(input("No. of Cannibals : "))

b = int(input("Boat size: "))

allpaths = []

def is\_valid(state):

m1, c1, n = state

m2 = m - m1

c2 = c - c1

if m1 < 0 or m2 < 0 or c1 < 0 or c2 < 0:

return False

if (m1 and m1 < c1) or (m2 and m2 < c2):

return False

return True

def generate\_successors(state):

m, c, n = state

successors = []

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

for action in actions:

moved\_ms, moved\_cn =action

if n == 1:

new\_state = (m - moved\_ms, c - moved\_cn, 0)

else:

new\_state = (m + moved\_ms, c + moved\_cn, 1)

if is\_valid(new\_state):

successors.append(new\_state)

return successors

def bfs():

start\_state = (m, c, 1)

goal\_state = (0, 0, 0)

visited = set()

q = deque([(start\_state,[])])

while q:

current\_state = q.popleft()

state, path = current\_state

if state in visited:

continue

path.append(state)

if state == goal\_state:

allpaths.append(path)

continue

visited.add(state)

for successor in generate\_successors(state):

if successor not in visited:

q.append((successor,path.copy()))

bfs()

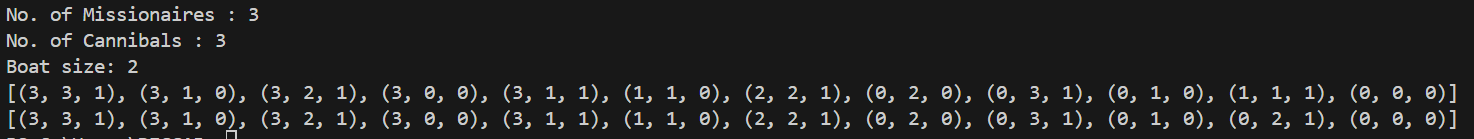
if len(allpaths)==0:

print("No Solutions")

else:

for p in allpaths:

Output:



2]Missionary and Cannibals Using DFS

from queue import deque

m = int(input("No. of Missionaires : "))

c = int(input("No. of Cannibals : "))

b = int(input("Boat size: "))

allpaths = []

def is\_valid(state):

m1, c1, n = state

m2 = m - m1

c2 = c - c1

if m1 < 0 or m2 < 0 or c1 < 0 or c2 < 0:

return False

if (m1 and m1 < c1) or (m2 and m2 < c2):

return False

return True

def generate\_successors(state):

m, c, n = state

successors = []

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

for action in actions:

moved\_ms, moved\_cn =action

if n == 1:

new\_state = (m - moved\_ms, c - moved\_cn, 0)

else:

new\_state = (m + moved\_ms, c + moved\_cn, 1)

if is\_valid(new\_state):

successors.append(new\_state)

return successors

def dfs():

start\_state = (m, c, 1)

goal\_state = (0, 0, 0)

visited = set()

q = deque([(start\_state,[])])

while q:

current\_state = q.pop()

state, path = current\_state

if state in visited:

continue

path.append(state)

if state == goal\_state:

allpaths.append(path)

continue

visited.add(state)

for successor in generate\_successors(state):

if successor not in visited:

q.append((successor,path.copy()))

dfs()

if len(allpaths)==0:

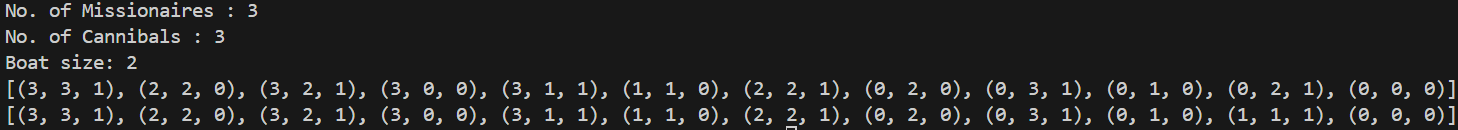
print("No Solutions")

else:

for p in allpaths:

print(p)

Output :



Conclusion: Solved Missionary and Cannibal Problem using BFS and DFS search algorithms with successful execution of programs.