**Department of Computer Engineering**

**Academic Term II: 23-24**

**Class: B.E (Computer), Sem – VI Subject Name: Artificial Intelligence Student Name: Roll No: 9538**

| **Practical No:** | **3** |
| --- | --- |
| **Title:** | Use DFS problem solving method for  a) Water Jug Problem  b) Missionaries & Cannibals |
| **Date of Performance:** |  |
| **Date of Submission:** |  |

**Rubrics for Evaluation:**

| **Sr.**  **No** | **Performance Indicator** | **Excellent** | **Good** | **Below**  **Average** | **Marks** |
| --- | --- | --- | --- | --- | --- |
| 1 | On time Completion &  Submission (01) | 01 (On  Time) | NA | 00 (Not on  Time) |  |
| 2 | Logic/Algorithm Complexity analysis (03) | 03(Correct ) | 02(Partial) | 01 (Tried) |  |
| 3 | Coding Standards (03):  Comments/indention/Naming conventions  Test Cases /Output | 03(All  used) | 02 (Partial) | 01 (rarely  followed) |  |
| 4 | Post Lab Assignment (03) | 03(done  well) | 2 (Partially  Correct) | 1(submitte  d) |  |
| **Total** | | | | |  |

**Signature of the Teacher:**

a)Water Jug

def pour\_water(state, action):

x, y = state

if action == 'fill\_4':

return (4, y)

elif action == 'fill\_3':

return (x, 3)

elif action == 'empty\_4':

return (0, y)

elif action == 'empty\_3':

return (x, 0)

elif action == 'pour\_4\_to\_3':

amount = min(x, 3 - y)

return (x - amount, y + amount)

elif action == 'pour\_3\_to\_4':

amount = min(y, 4 - x)

return (x + amount, y - amount)

else:

return state

def dfs(state, visited):

if state[0] == 2:

return [state]

visited.add(state)

for action in ['fill\_4', 'fill\_3', 'empty\_4', 'empty\_3', 'pour\_4\_to\_3', 'pour\_3\_to\_4']:

new\_state = pour\_water(state, action)

if new\_state not in visited:

path = dfs(new\_state, visited)

if path:

return [state] + path

return None

def print\_steps(path):

for i, state in enumerate(path):

jug\_4, jug\_3 = state

print(f"Step {i+1}: Jug 4: {jug\_4} gallons, Jug 3: {jug\_3} gallons")

initial\_state = (0, 0)

visited = set()

path = dfs(initial\_state, visited)

if path:

print("Steps to measure 2 gallons:")

print\_steps(path)

else:

print("No solution found.")

Output:

Steps to measure 2 gallons:

Step 1: Jug 4: 0 gallons, Jug 3: 0 gallons

Step 2: Jug 4: 4 gallons, Jug 3: 0 gallons

Step 3: Jug 4: 4 gallons, Jug 3: 3 gallons

Step 4: Jug 4: 0 gallons, Jug 3: 3 gallons

Step 5: Jug 4: 3 gallons, Jug 3: 0 gallons

Step 6: Jug 4: 3 gallons, Jug 3: 3 gallons

Step 7: Jug 4: 4 gallons, Jug 3: 2 gallons

Step 8: Jug 4: 0 gallons, Jug 3: 2 gallons

Step 9: Jug 4: 2 gallons, Jug 3: 0 gallons

b) Missionaries & Cannibals

class State:

def \_\_init\_\_(self, missionaries, cannibals, boat\_position):

self.missionaries = missionaries

self.cannibals = cannibals

self.boat\_position = boat\_position

def is\_valid(self):

if (

0 <= self.missionaries <= 3

and 0 <= self.cannibals <= 3

and 0 <= self.boat\_position <= 1

):

if (

self.missionaries == 0

or self.missionaries == 3

or self.missionaries >= self.cannibals

):

return True

return False

def is\_goal(self):

return self.missionaries == 0 and self.cannibals == 0 and self.boat\_position == 0

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

return (

self.missionaries == other.missionaries

and self.cannibals == other.cannibals

and self.boat\_position == other.boat\_position

)

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

return hash((self.missionaries, self.cannibals, self.boat\_position))

def generate\_next\_states(current\_state):

next\_states = []

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

for m, c in moves:

if current\_state.boat\_position == 1:

new\_state = State(

current\_state.missionaries - m,

current\_state.cannibals - c,

0,

)

else:

new\_state = State(

current\_state.missionaries + m,

current\_state.cannibals + c,

1,

)

if new\_state.is\_valid():

next\_states.append(new\_state)

return next\_states

def dfs\_search():

start\_state = State(3, 3, 1)

goal\_state = State(0, 0, 0)

stack = [(start\_state, [])]

visited = set()

while stack:

current\_state, path = stack.pop()

if current\_state.is\_goal():

return path

if current\_state not in visited:

visited.add(current\_state)

next\_states = generate\_next\_states(current\_state)

for next\_state in next\_states:

if next\_state not in visited:

stack.append((next\_state, path + [current\_state]))

return None

def print\_state\_description(state):

left\_shore = f"{state.missionaries} Missionaries and {state.cannibals} Cannibals on the Left Shore"

right\_shore = f"{3 - state.missionaries} Missionaries and {3 - state.cannibals} Cannibals on the Right Shore"

print(f"{left\_shore}, {right\_shore}\n")

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

solution\_path = dfs\_search()

if solution\_path:

print("Solution Path:")

for i, state in enumerate(solution\_path):

print(f"Step {i + 1}:")

print\_state\_description(state)

else:

print("No solution found.")

Output:

Solution Path:

Step 1:

3 Missionaries and 3 Cannibals on the Left Shore, 0 Missionaries and 0 Cannibals on the Right Shore

Step 2:

2 Missionaries and 2 Cannibals on the Left Shore, 1 Missionaries and 1 Cannibals on the Right Shore

Step 3:

3 Missionaries and 2 Cannibals on the Left Shore, 0 Missionaries and 1 Cannibals on the Right Shore

Step 4:

2 Missionaries and 1 Cannibals on the Left Shore, 1 Missionaries and 2 Cannibals on the Right Shore

Step 5:

2 Missionaries and 2 Cannibals on the Left Shore, 1 Missionaries and 1 Cannibals on the Right Shore

Step 6:

1 Missionaries and 1 Cannibals on the Left Shore, 2 Missionaries and 2 Cannibals on the Right Shore

Step 7:

3 Missionaries and 1 Cannibals on the Left Shore, 0 Missionaries and 2 Cannibals on the Right Shore