Name: Mansi Dhamne
UID: 2022302140

Artificial Intelligence and Machine Learning
Experiment 2

Problem Statement: Implement a given problem using the Uninformed (DFS) searching technique. Analyze the algorithms with respect to completeness, optimality, time and space complexity.

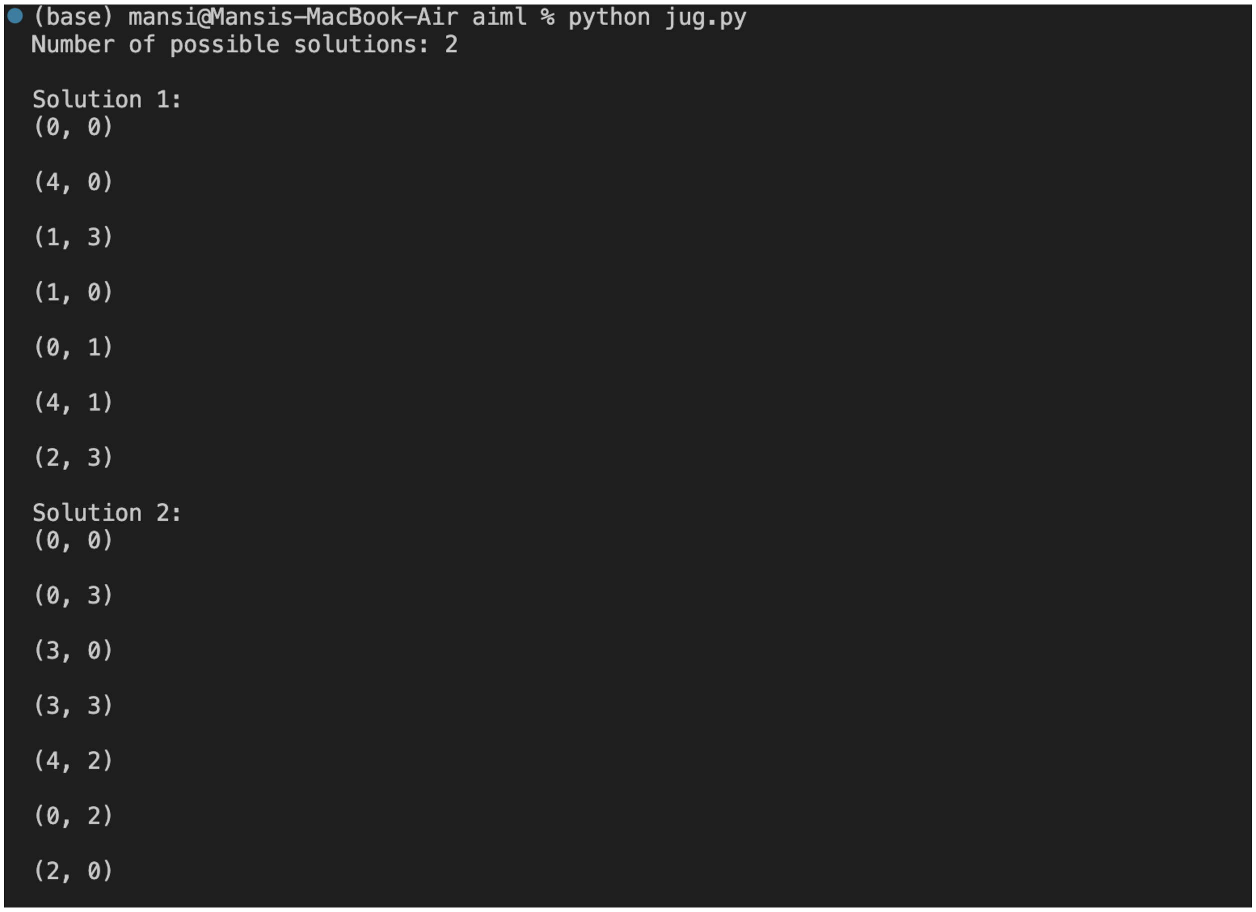
Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in a 4-gallon jug?

Production Rules for Water Jug Problem in Artificial Intelligence
1 (x, y) is X<4 ->(4, Y) Fill the 4-liter jug
2 (x, y) if Y<3 -> (x, 3) Fill the 3-liter jug
3 (x, y) if x>0 -> (x-d, d) Pour some water out of the 4-liter jug.
4 (x, y) if Y>0 -> (d, y-d) Pour some water out of the 3-liter jug.
5 (x, y) if x>0 -> (0, y) Empty the 4-liter jug on the ground
6 (x, y) if y>0 -> (x,0) Empty the 3-liter jug on the ground
7 (x, y) if X+Y >= 4 and y>0 -> (4, y-(4-x))
Pour water from the 3-liter jug into the 4-liter jug until the 4-liter jug is full
8 (x, y) if X+Y>=3 and x>0 -> (x-(3-y),3))
Pour water from the 4-liter jug into the 3-liter jug until the 3-liter jug is full.
9 (x, y) if X+Y <=4 and y>0 -> (x+y, 0) Pour all the water from the 3-liter jug into the 4-liter jug.
10 (x, y) if X+Y<=3 and x>0 -> (0, x+y) Pour all the water from the 4-liter jug into the 3-liter jug.

Implementation:

```python
from collections import deque

def bfs():
   start_state = (0, 0)
   queue = deque([(start_state, [])])
   visited = set()
   all_solutions = []

   while queue:
       current_state, path = queue.popleft()
       x, y = current_state
       if x == 2:
           all_solutions.append(path + [current_state])
       visited.add(current_state)
       next_states = [
           (4, y),
           (x, 3),
           (0, y),
           (x, 0),
           (x - min(x, 3 - y), y + min(x, 3 - y)),
           (x + min(y, 4 - x), y - min(y, 4 - x)),
       ]

       for next_state in next_states:
           if next_state not in visited:
               queue.append((next_state, path + [current_state]))
               visited.add(next_state)

   return all_solutions

solutions = bfs()
if solutions:
   print(f"Number of possible solutions: {len(solutions)}\n")
   for i, solution in enumerate(solutions, 1):
       print(f"Solution {i}:")
       for step in solution:
           print(step)
           print()
else:
   print("No solution found.")
```

Solving:



```
(base) mansi@Mansis-MacBook-Air aiml % python jug.py
Number of possible solutions: 2

Solution 1:
(0, 0)

(4, 0)

(1, 3)

(1, 0)

(0, 1)

(4, 1)

(2, 3)

Solution 2:
(0, 0)

(0, 3)

(3, 0)

(3, 3)

(4, 2)

(0, 2)

(2, 0)
```

Analysis of the BFS Algorithm:

1. Completeness:

   BFS is complete, meaning it **will always find a solution if one exists**.
   In this problem, BFS will explore all possible states, ensuring it finds a path to the goal (2 gallons in the 4-gallon jug)

2. Optimality:

   BFS is optimal when **all actions have the same cost**, as it finds the shortest path in an unweighted graph.
   In this case, each state transition is considered to have the same "cost," so BFS will yield the optimal (shortest) solution.

3. Time Complexity:

   The time complexity of BFS is **O(b^d)**, where **b is the branching factor** (number of possible moves from each state), and **d is the depth of the shallowest solution**.
   Since this problem has a finite number of states, BFS will explore each state at most once. Thus, the time complexity is proportional to the number of states.

4. Space Complexity:

The space complexity of BFS is also **O(b^d)** because it **stores all generated states** in memory. The primary memory usage comes from **storing the queue** of nodes to explore and the **set of visited states**.

Conclusion: BFS is a robust and effective method for solving the Water Jug Problem. It guarantees finding the shortest sequence of moves (if any exist) to get exactly 2 gallons of water in the 4-gallon jug. The algorithm is both complete and optimal for this problem, with time and space complexities that are manageable for this finite state space.