

EX.NO:3

DATE:16/10/2024

Reg.no:220701010

DEPTH-FIRST SEARCH – WATER JUG PROBLEM

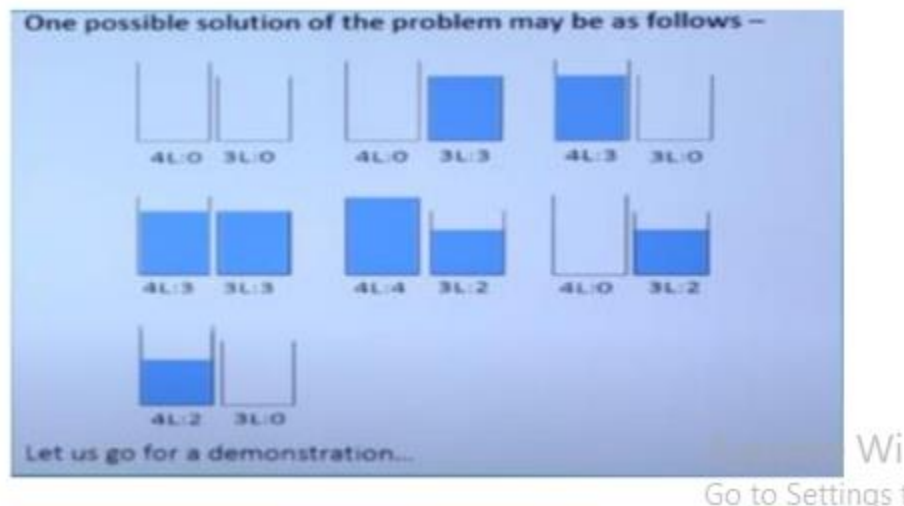
In the water jug problem in Artificial Intelligence, we are provided with two jugs: one having

the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water.

There is no other measuring equipment available and the jugs also do not have any kind of marking

on them. So, the agent's task here is to fill the 4-gallon jug with 2 gallons of water by using only

these two jugs and no other material. Initially, both our jugs are empty.



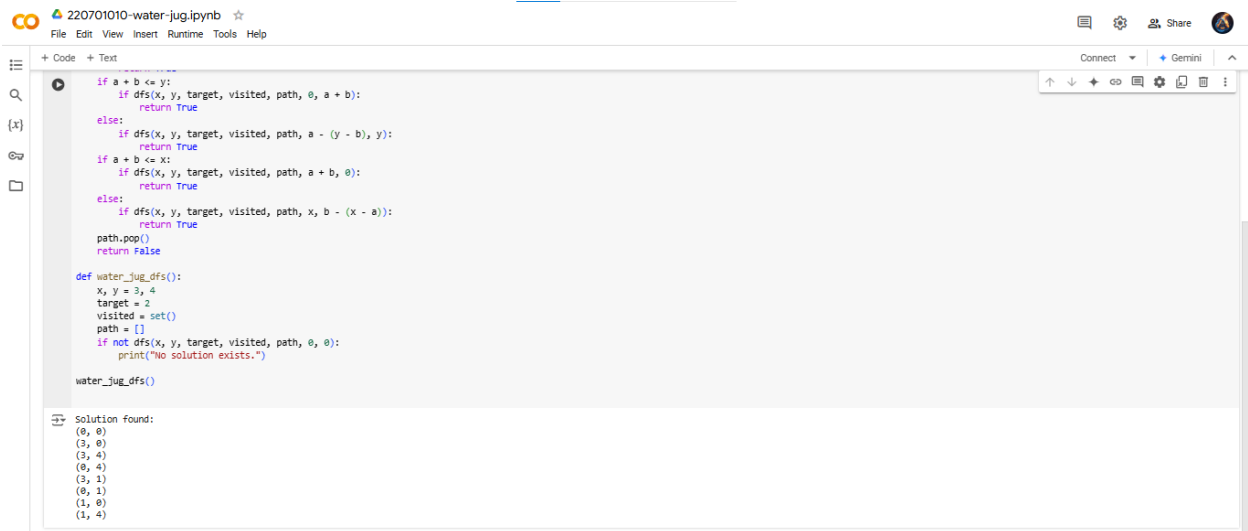
CODE:

```
def dfs(x, y, target, visited, path, a, b):
    if b == target:
        print("Solution found:")
        for step in path:
            print(step)
        return True
    if (a, b) in visited:
        return False
    visited.add((a, b))
    path.append(f"({a}, {b})")
    if dfs(x, y, target, visited, path, x, b):
        return True
    if dfs(x, y, target, visited, path, a, y):
        return True
    if dfs(x, y, target, visited, path, 0, b):
        return True
    if dfs(x, y, target, visited, path, a, 0):
        return True
    if a + b <= y:
        if dfs(x, y, target, visited, path, 0, a + b):
            return True
    else:
        if dfs(x, y, target, visited, path, a - (y - b), y):
            return True
    if a + b <= x:
        if dfs(x, y, target, visited, path, a + b, 0):
            return True
    else:
        if dfs(x, y, target, visited, path, x, b - (x - a)):
            return True
    path.pop()
    return False

def water_jug_dfs():
    x, y = 3, 4
    target = 2
    visited = set()
    path = []
    if not dfs(x, y, target, visited, path, 0, 0):
        print("No solution exists.")

water_jug_dfs()
```

OUTPUT:



The screenshot shows a Jupyter Notebook titled "220701010-water-jug.ipynb". The code implements a Depth-First Search (DFS) algorithm to solve the water jug problem. The algorithm starts with two jugs, X and Y, both empty (0, 0). The goal is to reach a state where the total water in both jugs equals the target (2). The code defines a recursive function `dfs(x, y, target, visited, path, a, b)` that explores all possible states. The states are represented as (X, Y) pairs. The code also includes a `water_jug_dfs()` function that initializes the variables and calls the `dfs` function. The output of the code is a list of states that lead to the solution, starting from (0, 0) and ending at (1, 4).

```
def dfs(x, y, target, visited, path, a, b):
    if a + b <= y:
        if dfs(x, y, target, visited, path, a, a + b):
            return True
    else:
        if dfs(x, y, target, visited, path, a - (y - b), y):
            return True
    if a + b <= x:
        if dfs(x, y, target, visited, path, a + b, 0):
            return True
    else:
        if dfs(x, y, target, visited, path, x, b - (x - a)):
            return True
    path.pop()
    return False

def water_jug_dfs():
    X, Y = 3, 4
    target = 2
    visited = set()
    path = []
    if not dfs(X, Y, target, visited, path, 0, 0):
        print("No solution exists.")
    water_jug_dfs()
```

Solution found:

- (0, 0)
- (3, 0)
- (3, 4)
- (0, 4)
- (3, 1)
- (0, 1)
- (1, 0)
- (1, 4)