

EX.NO:3

DATE:16/10/2024

Reg.no:220701026

DEPTH-FIRST SEARCH – WATER JUG PROBLEM

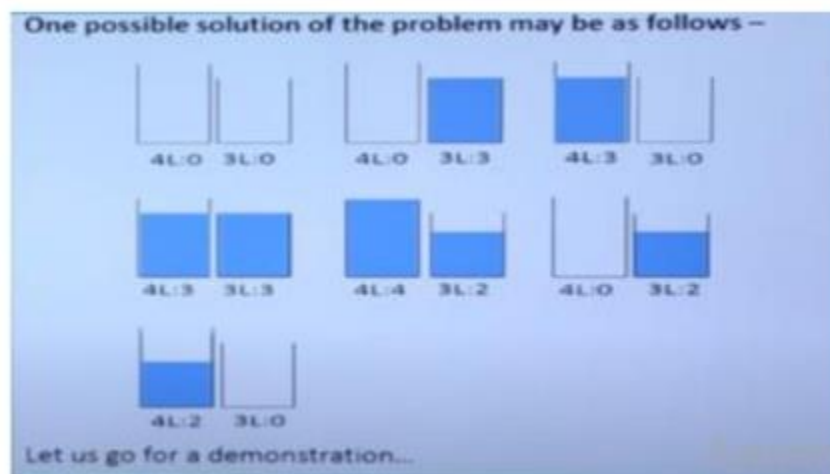
In the water jug problem in Artificial Intelligence, we are provided with two jugs: one having

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.



Wi
Go to Settings 1

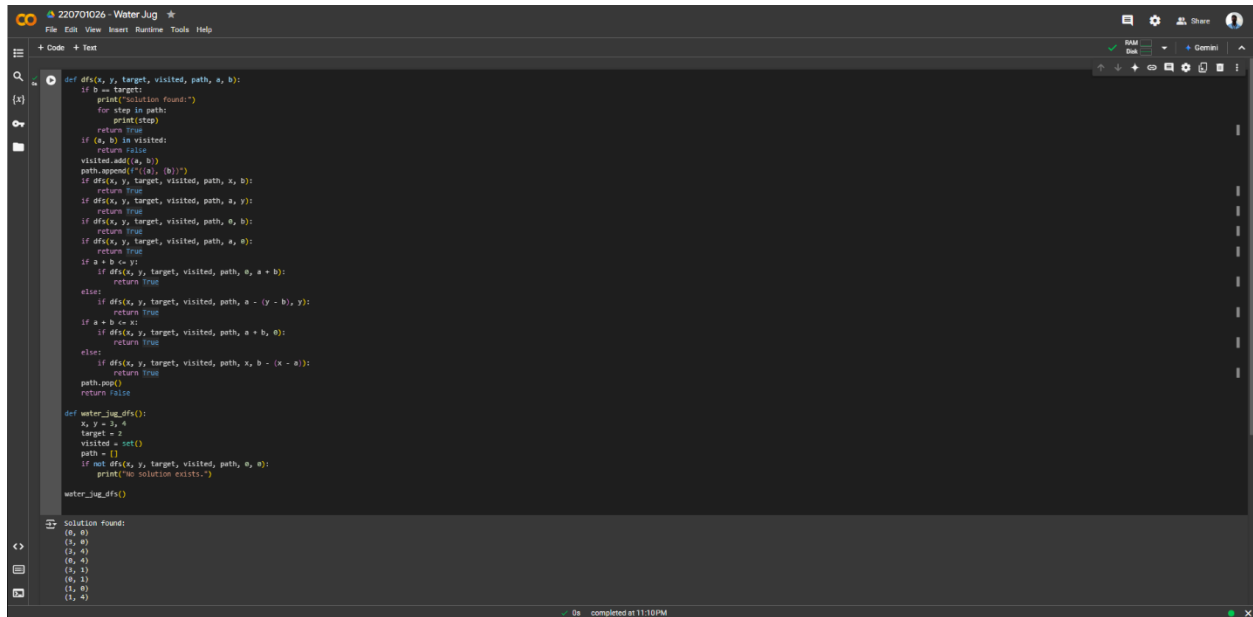
CODE:

```
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 code editor window titled "220701026 - Water Jug". The code implements a Depth-First Search (DFS) algorithm to solve the Water Jug problem. The main function, `water_jug_dfs()`, initializes variables `x = 3`, `y = 4`, and `target = 5`, sets a `visited` set, and calls the `dfs` function. The `dfs` function recursively explores all possible states of the jugs, checking for the target state. The output at the bottom shows the solution found: `(0, 0)`, `(3, 0)`, `(2, 4)`, `(0, 4)`, `(0, 1)`, `(3, 1)`, `(0, 3)`, and `(3, 4)`.

```
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("({a}, {b})")
    if dfs(x, y, target, visited, path, a, 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, 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 = 5
    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)
(2, 4)
(0, 4)
(0, 1)
(3, 1)
(0, 3)
(3, 4)
```

completed at 11:10 PM