SUPERIOR UNIVERSITY

Submitted By:

Name: **Ali Raza**

Roll number **Su92-bsdsm-f23-018**

Section: **4A**

Task: **03**

Subject: **Programming For AI (lab)**

Submitted to: **Sir Rasikh**

1. Function (dfs)

def dfs(jugA_capacity, jugB_capacity, target_amount):

This function uses **DFS (Depth-First Search)** to find a solution to the **Water Jug Problem**.

Parameters:

- jugA_capacity: The total capacity of Jug A.
- jugB_capacity: The total capacity of Jug B.
- target_amount: The target amount of water we need to measure.

---

2. Initializing Variables

stack = [(0, 0)]

visited = set()

actions = []

- stack: A **list** that acts as a **stack (LIFO)** to keep track of the states (amount of water in Jug A and Jug B).
- visited: A **set** to keep track of visited states, preventing infinite loops.
- actions: A **list** to store the sequence of steps leading to the solution.

---

3. DFS Algorithm Execution

while stack:

  jugA, jugB = stack.pop()

  actions.append((jugA, jugB))

- **Loop until the stack is empty**.
- **Pop the last state** (current amounts of water in both jugs).
- **Store the action** (amount of water in jugs at that step).

---

4. Check for Solution

if jugA == target_amount or jugB == target_amount:

  print("Solution Found")

  for action in actions:

```
        print(action)
    return True
```

- If **either jug contains the target amount of water**, print the solution steps.
- Return True to indicate a solution was found.

---

5. Mark the State as Visited

```
visited.add((jugA, jugB))
```

- Mark the **current state** as visited to avoid redundant checks.

---

6. Generate Possible Next States

```
states = [
    (jugA_capacity, jugB),  # Fill Jug A
    (jugA, jugB_capacity),  # Fill Jug B
    (0, jugB),              # Empty Jug A
    (jugA, 0),              # Empty Jug B
    (jugA - min(jugA, jugB_capacity - jugB), jugB + min(jugA, jugB_capacity - jugB)),  # Pour A → B
    (jugA + min(jugB, jugA_capacity - jugA), jugB - min(jugB, jugA_capacity - jugA)),  # Pour B → A
]
```

These are **all possible moves** in the problem:

1. **Fill Jug A** completely.
2. **Fill Jug B** completely.
3. **Empty Jug A**.
4. **Empty Jug B**.
5. **Pour water from Jug A into Jug B** (until Jug B is full or Jug A is empty).
6. **Pour water from Jug B into Jug A** (until Jug A is full or Jug B is empty).

---

7. Add Valid States to Stack

```
for state in states:
    if state not in visited:
        stack.append(state)
```

- Check if each possible state has been **visited before**.
- If not, **add it to the stack** for further exploration.

---

8. No Solution Case

```
print("No Solution Found")
return False
```

- If the loop exits and no solution is found, print "No Solution Found".

Executing the Function

```
jugA_capacity = 5
jugB_capacity = 3
target_amount = 2

dfs(jugA_capacity, jugB_capacity, target_amount)
```

- **Jug A capacity** = 5 liters
- **Jug B capacity** = 3 liters
- **Target amount** = 2 liters
- The function is called to find a **valid sequence of operations** to measure 2 liters of water.

Output

```
Solution Found
(0, 0)
(5, 0)
(2, 3)
```

- **Step 1:** Start with (0, 0).
- **Step 2:** Fill **Jug A** → (5, 0).

- **Step 3:** Pour **from Jug A into Jug B** → (2, 3), leaving **2 liters in Jug A** (solution found).

```python
def is_position_safe(board, row, col):
    for i in range(row):
        if board[i] == col or \
           board[i] - i == col - row or \
           board[i] + i == col + row:
            return False
    return True
def solve_n_queens_dynamic(size):
    def solve(row, board):
        if row == size:
            solutions.append(board[:])
            return
        for col in range(size):
            if is_position_safe(board, row, col):
                board[row] = col
                solve(row + 1, board)
                board[row] = -1
    solutions = []
    board = [-1] * size
    solve(0, board)
    return solutions
def display_n_queens_solutions(solutions):
    for solution in solutions:
        for i in range(len(solution)):
            row = ['.'] * len(solution)
            row[solution[i]] = 'Q'
            print(" ".join(row))
        print("\n")
board_size = 6
solutions = solve_n_queens_dynamic(board_size)
print(f"Number of solutions: {len(solutions)}")
display_n_queens_solutions(solutions)
```

```
Number of solutions: 4
. Q . . . .
. . . Q . .
. . . . . Q
Q . . . . .
. . Q . . .
. . . . Q .


. . Q . . .
. . . . . Q
. Q . . . .
. . . . Q .
Q . . . . .
. . . Q . .


. . . Q . .
Q . . . . .
. . . . Q .
. Q . . . .
. . . . . Q
. . Q . . .

...
. . . Q . .
. Q . . . .


Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```