Name : Anuska Nath
Dept : IT (UG3)
Section : A3
Roll : 002311001003

Assignment 3

Write a Python program to solve the water-jug problem using breadth-first search. Inputs to the program should be 1. Capacity of two jugs, and target measurement. Target measurement is the unit user wants to get as a solution.

Water Jug Problem – BFS Approach

Problem Objective:

Given two jugs with capacities capacity1 and capacity2, the goal is to measure exactly target units of water, ensuring that the target is stored in the jug with the larger capacity. For example, with input (4, 3, 2), we must get exactly 2 liters in Jug 1 (since 4 > 3).

Algorithm Used:

The solution uses Breadth-First Search (BFS) implemented in the function bfs_all_solutions(capacity1, capacity2, target).

- **Step 1 – Problem Setup**

    1. Start with two jugs: Jug 1 with capacity capacity1 and Jug 2 with capacity capacity2.
    2. The target amount must be stored in the larger jug at the end.
    3. Represent the current state of the problem as a tuple (x, y), where:
        - x = amount of water in Jug 1
        - y = amount of water in Jug 2

- **Step 2 – BFS Initialization**
    - Use a queue (queue) to store states to be explored, starting with (0, 0) (both jugs empty).
    - Maintain a visited set (visited) to avoid processing the same state twice.
    - Maintain a parent mapping (parents) to store all predecessor states for each state (needed to reconstruct multiple solutions).
    - Keep a level dictionary (level) to store BFS depth for each state so we know when the shortest path level is reached.
- **Step 3 – BFS State Expansion**
  For each state (x, y), generate all possible valid next states by:
    1. Filling Jug 1 → (capacity1, y)
    2. Filling Jug 2 → (x, capacity2)
    3. Emptying Jug 1 → (0, y)
    4. Emptying Jug 2 → (x, 0)
    5. Pouring from Jug 1 to Jug 2 until Jug 1 is empty or Jug 2 is full
    6. Pouring from Jug 2 to Jug 1 until Jug 2 is empty or Jug 1 is full
- **Step 4 – Goal Condition**
    - If capacity1 >= capacity2 → check if x == target (target in Jug 1).
    - Else → check if y == target (target in Jug 2).
    - Once a goal state is found, store it in goal_states and record the BFS level.
    - Continue exploring only states at the same BFS level to ensure we find all shortest solutions.

- **Step 5 – Backtracking to Find All Solutions**
    - For each goal state, use stack-based backtracking starting from the goal and moving backward using parents.
    - Since parents[state] may have multiple entries, this process generates all possible shortest solution paths.
    - Reverse each path to get the correct order from (0, 0) to the goal state.
- **Step 6 – Output**
    - Print the total number of solutions and each solution path to the console.
    - Save the results to water_jug_solutions.txt, including:
        - Total solutions found
        - Which jug holds the target
        - Numbered solution paths in (jug1, jug2) format

Execution Example :

Input:

capacity1 = 4

capacity2 = 3

target = 2

Output (Console):

Total solutions found: 2 (Target in Jug 1)

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)

All solutions have been saved to 'output.txt'

Output File:

All solutions are also saved in water_jug_solutions.txt, formatted with total solution count, target jug information, and numbered solution paths.

Conclusion:

The BFS-based approach efficiently explores all possible states of the two jugs to find every valid sequence of moves that leads to the target measurement in the larger jug. By systematically generating and tracking states, the algorithm guarantees the discovery of all shortest solutions, which are clearly presented in both console output and the generated solution file for documentation.