**Experiment:1**

**Aim: Write a Program to Implement Water-Jug problem using Breadth First Search algorithm.**

**Description of water jug problem:**

The water Jug Problem, as the name suggests, is a problem where two jugs of water are given, say one is a 4-litre one, and the other one is a 3-litre one, but none of the measuring markers is mentioned on any of it. There is a pump available to fill the jugs with water. How can you exactly pour 2 litres of water into a 4-litre jug? Assuming that both the jugs are empty, the task is to find a solution to pour 2-litre water into a 4-litre jug.

**Solution of water jug problem:**

State: (x, y), where x represents the quantity of water in a 4-liter jug and y represents the quantity of water in a 3-liter jug. That is, x = 0, 1, 2, 3, or 4  y = 0, 1, 2, 3

*Start state: (0, 0).*

*Goal state: (2, n) for any n.*

Here need to start from the current state and end up in a goal state.

**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. |
| 11 | (0, 2) -> (2, 0) | Pour the 2-liter water from the 3-liter jug into the 4-liter jug. |
| 12 | (2, Y) -> (0, y) | Empty the 2-liter in the 4-liter jug on the ground. |

**The solution to Water Jug Problem in Artificial Intelligence**

1. Current state = (0, 0)

2. Loop until the goal state (2, 0) reached

  – Apply a rule whose left side matches the current state

  – Set the new current state to be the resulting state

(0, 0) – Start State

(0, 3) – Rule 2, Fill the 3-liter jug

(3, 0) – Rule 9, Pour all the water from the 3-liter jug into the 4-liter jug.

(3, 3) – Rule 2, Fill the 3-liter jug

(4, 2) – Rule 7, Pour water from the 3-liter jug into the 4-liter jug until the 4-liter jug is full.

(0, 2) –  Rule 5, Empty the 4-liter jug on the ground

(2, 0) – Rule 9, Pour all the water from the 3-liter jug into the 4-liter jug.

Goal State reached

**Another solution to Water Jug Problem in Artificial Intelligence**

(0, 0) – Start State

(4, 0) – Rule 1, Fill the 4-liter jug

(1, 3) – Rule 8, Pour water from the 4-liter jug into the 3-liter jug until the 3-liter jug is full.

(1, 0) – Rule 6, Empty the 3-liter jug on the ground

(0, 1) – Rule 10, Pour all the water from the 4-liter jug into the 3-liter jug.

(4, 1) –  Rule 1, Fill the 4-liter jug

(2, 3) – Rule 8, Pour water from the 4-liter jug into the 3-liter jug until the 3-liter jug is full.

Goal State reached

**Implementation:**

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a ‘search key’) and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level. It is implemented using a queue.

**Implementation of BFS in Water Jug problem-:**

Here in this problem using BFS initial state is (0,0) and Redundant case is (4,3). So, in BFS we skip initial and Redundant case as it is already visited and from (4,3) we can go to many possible states which can be takes as nodes to traverse.

**from** collections **import** deque

**def** BFS(a, b, target):

    m **=** {}

    isSolvable **=** False

    path **=** []

    q **=** deque()

    q.append((0, 0))

**while** (len(q) > 0):

        u **=** q.popleft()# If this state is already visited

**if** ((u[0], u[1]) **in** m):

**continue**

**if** ((u[0] > a **or** u[1] > b **or**

             u[0] < 0 **or** u[1] < 0)):

**continue**

        # Filling the vector for constructing

        # the solution path

        path.append([u[0], u[1]])

        # Marking current state as visited

        m[(u[0], u[1])] **=** 1

        # If we reach solution state, put ans=1

**if** (u[0] **==** target **or** u[1] **==** target):

            isSolvable **=** True

**if** (u[0] **==** target):

**if** (u[1] !**=** 0):

                    # Fill final state

                    path.append([u[0], 0])

**else**:

**if** (u[0] !**=** 0):

                    # Fill final state

                    path.append([0, u[1]])

            # Print the solution path

            sz **=** len(path)

**for** i **in** range(sz):

                print("(", path[i][0], ",",

                      path[i][1], ")")

**break**

        # If we have not reached final state

        # then, start developing intermediate

        # states to reach solution state

        q.append([u[0], b])  # Fill Jug2

        q.append([a, u[1]])  # Fill Jug1

**for** ap **in** range(max(a, b) **+** 1):

            # Pour amount ap from Jug2 to Jug1

            c **=** u[0] **+** ap

            d **=** u[1] **-** ap

            # Check if this state is possible or not

**if** (c **==** a **or** (d **==** 0 **and** d >**=** 0)):

                q.append([c, d])

            # Pour amount ap from Jug 1 to Jug2

            c **=** u[0] **-** ap

            d **=** u[1] **+** ap

            # Check if this state is possible or not

**if** ((c **==** 0 **and** c >**=** 0) **or** d **==** b):

                q.append([c, d])

        # Empty Jug2

        q.append([a, 0])

        # Empty Jug1

        q.append([0, b])

    # No, solution exists if ans=0

**if** (**not** isSolvable):

**print**("No solution")

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    Jug1, Jug2, target **=** 4, 3, 2

**print**("Path from initial state "

          "to solution state ::")

    BFS(Jug1, Jug2, target)

**Output**

Path of states of jugs followed is :

0 , 0

0 , 3

3 , 0

3 , 3

4 , 2

0 , 2

**Date of experiment performed: 16/01/2024**

**Day of experiment performed: Tuesday**

**Date of experiment submission: 23/01/2024**

**Day of experiment Submission: Tuesday**

Faculty Co-ordinator Signature