Experiment No : 2 Date:

Water Jug Problem

AIM: To solve the Water Jug problem using BFS and DFS algorithms.

THEORY:

Problem Statement:

The Water Jug Problem is a classic puzzle where you are given a set of jugs with different capacities and tasked with measuring a specific quantity of water using these jugs. The goal is to find a sequence of pouring actions that will result in obtaining the desired quantity of water.

State Space:

The state space for the Water Jug Problem consists of all possible configurations of water in the jugs. Each state is represented by the amount of water present in each jug. The state space is constrained by the capacities of the jugs.

The state space for this problem can be described as the set of ordered pairs of integers (x, y) such that $x \in \{0,1,2\ldots,a\}$ and $y \in \{0,1,2\ldots,b\}$, where a and b are the maximum permissible limits of each jug. The initial state is (0,0), and the goal states are (Goal, y) and (x, Goal) where Goal is the desired quantity.

Production Rules:

- Fill Rule: Fill a jug to its maximum capacity.
- Empty Rule: Empty a jug completely.
- Pour Rule: Pour water from one jug into another until either the source jug is empty or the destination jug is full.

Algorithms:

a) BFS-ALGORITHM:

```
BFS()
Open=((start, Nil)), closed=()
While Open is not empty do
   Nodepair = head(open)
   node = head(Nodepair)
   if Goalstate(node) == True then
      Return reconstructpath(Nodepair, closed)
   else
```

```
    closed = cons(Nodepair, closed)
    children = movegen(node)
    noloop = removeseen(children, open, closed)
    new = makepair(noloop, node)
    Open = Append(tail(Open), new)
Return Failure
```

b) DFS-ALGORITHM:

```
DFS()
Open=((start, Nil)), closed=()
While Open is not empty do
  Nodepair = head(open)
  node = head(Nodepair)
  if Goalstate(node) == True then
    Return reconstructpath(Nodepair, closed)
  else
    closed = cons(Nodepair, closed)
    children = movegen(node)
    noloop = removeseen(children, open, closed)
    new = makepair(noloop, node)
    Open = Append(new, tail(Open))
Return Failure
```

Example : Suppose we have two jugs with capacities of 3 liters and 5 liters, and we need to measure exactly 4 liters of water. Step-by-step solution:

Start with both jugs empty: (0, 0)

Fill the 5-liter jug: (0, 5)

Pour water from the 5-liter jug into the 3-liter jug: (3, 2)

Empty the 3-liter jug: (0, 2)

Pour the remaining 2 liters from the 5-liter jug into the 3-liter jug: (2, 0)

Fill the 5-liter jug again: (2, 5)

Pour water from the 5-liter jug into the 3-liter jug until the 3-liter jug is full: (3, 4)

CODE: **1) Water Jug Using BFS**

```
from collections import deque
def WaterJugBFS(goal,Left_Capacity,Right_Capacity):
    visited = set()
    queue = deque([(0, 0, [(0, 0)])])  #curren state and path
    while queue:
        Left, Right, path = queue.popleft()
        if Left == goal or Right == goal:
            allpath.append(path)
            continue
        visited.add((Left,Right))
        succ = [(Left_Capacity, Right), (Left, Right_Capacity), (0, Right), (Left, 0)]
        SpaceLeft = Left_Capacity - Left
        if SpaceLeft >= Right:
            succ.append((Left + Right, 0))
        else:
            succ.append((Left_Capacity, Right - SpaceLeft))


        SpaceRight = Right_Capacity - Right
        if SpaceRight >= Left:
            succ.append((0, Left + Right))
        else:
            succ.append((Left - SpaceRight, Right_Capacity))
        for s in succ:
            if s not in visited:
                queue.append((s[0], s[1],path + [s] )) #left jug , right jug , path
allpath = []
print("Water Jug Problem (BFS)")
goal = int(input("Enter Goal "))
Left_Capacity = int(input("Enter Capacity of Left Jug : "))
Right_Capacity = int(input("Enter Capacity of Right Jug : "))
WaterJugBFS(goal,Left_Capacity,Right_Capacity)
if len(allpath)==0:
    print("No Solutions")
else:
    for p in allpath:
        print(p)
```

Output:

```
Water Jug Problem (BFS)
Enter Goal 2
Enter Capacity of Left Jug : 4
Enter Capacity of Right Jug : 3
[(0, 0), (0, 3), (3, 0), (3, 3), (4, 2)]
[(0, 0), (4, 0), (1, 3), (1, 0), (0, 1), (4, 1), (2, 3)]
```

2) Water Jug Using DFS

```
from collections import deque

def WaterJugDFS(goal,Left_Capacity,Right_Capacity):
   visited = set()
   stack = deque([(0, 0, [(0, 0)])])  #curren state and path
   while stack:
      Left, Right, path = stack.pop()

      if Left == goal or Right == goal:
         allpath.append(path)
         continue
      visited.add((Left,Right))
      succ = [(Left_Capacity, Right), (Left, Right_Capacity), (0, Right), (Left, 0)]
      SpaceLeft = Left_Capacity - Left

      if SpaceLeft >= Right:
         succ.append((Left + Right, 0))
      else:
         succ.append((Left_Capacity, Right - SpaceLeft))


      SpaceRight = Right_Capacity - Right
      if SpaceRight >= Left:
         succ.append((0, Left + Right))
      else:
         succ.append((Left - SpaceRight, Right_Capacity))
      for s in succ:
         if s not in visited:
            stack.append((s[0], s[1],path + [s] )) #left jug , right jug , path
allpath = []
print("Water Jug Problem (DFS)")

goal = int(input("Enter Goal "))
Left_Capacity = int(input("Enter Capacity of Left Jug : "))
Right_Capacity = int(input("Enter Capacity of Right Jug : "))
WaterJugDFS(goal,Left_Capacity,Right_Capacity)

if len(allpath)==0:
   print("No Solutions")
else:
    for p in allpath:
      print(p)
```

Output:

```
Water Jug Problem (DFS)
Enter Goal 2
Enter Capacity of Left Jug : 4
Enter Capacity of Right Jug : 3
[(0, 0), (0, 3), (3, 0), (3, 3), (4, 2)]
[(0, 0), (0, 3), (3, 0), (3, 3), (4, 3), (4, 0), (1, 3), (1, 0), (0, 1), (4, 1), (2, 3)]
```

Conclusion: Solved Water jug problem using BFS and DFS search algorithms with successful execution of programs.