**EXPERIMENT NO: 02 DATE: 13/02/24**

**Aim:** To implement Water Jug Problem using BFS and DFS search algorithms.

**Problem Statement:**

The Water Jug problem can be stated as: “Given two unmarked jugs having capacities ‘a’

and ‘b’ litres respectively and target volume ‘t’ litres find the moves that get exactly ‘t’

litres, in any of the two jugs.”

**Theory:**

* 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.

**State Space:**

The state space for this problem can be described as the set of ordered pair of

integers (x,y) such that x ε {0,1,2…,a} and y ε {0,1,2…,b}. The initial state is (0,0)

and the goal states are (t,y) and (x,t) for all x,y.

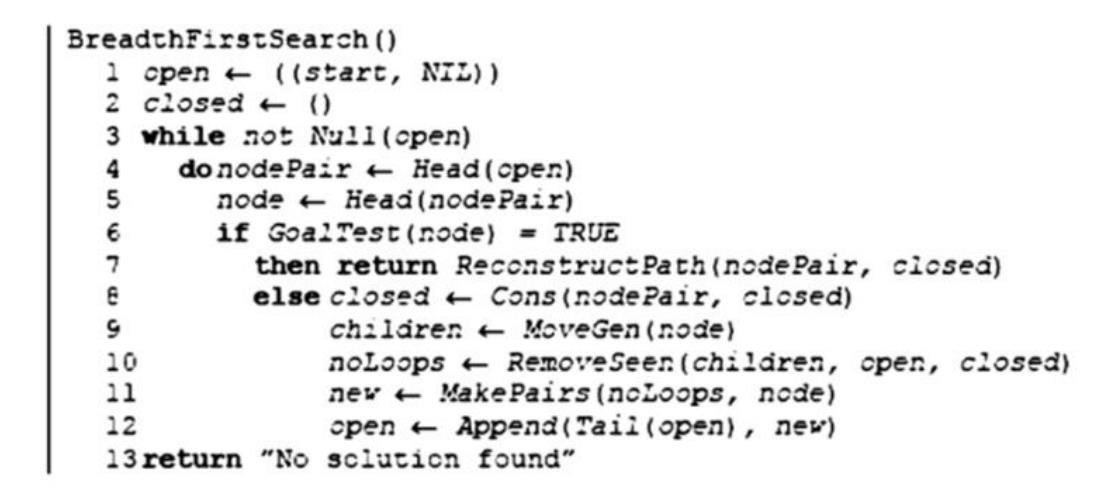
**Production Rules:**

Here, let ***x*** denote the 4-gallon jug and ***y*** denote the 3-gallon jug.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | **Initial State** | **Condition** | **Final state** | **Description of action taken** |
| 1. | (x,y) | If x<4 | (4,y) | Fill the 4 gallon jug completely. |
| 2. | (x,y) | if y<3 | (x,3) | Fill the 3-gallon jug completely. |
| 3. | (x,y) | If x>0 | (x-d,y) | Pour some part from the 4 gallon jug. |
| 4. | (x,y) | If y>0 | (x,y-d) | Pour some part from the 3 gallon jug. |
| 5. | (x,y) | If x>0 | (0,y) | Empty the 4 gallon jug. |
| 6. | (x,y) | If y>0 | (x,0) | Empty the 3 gallon jug. |

**a) Solution using BFS:**

**Algorithm:**



**Code:**

#water jug problem using BFS

visited = []

traversal = []

parent\_state = {}

A = int(input("Enter the maximum capacity of container A: "))

B = int(input("Enter the maximum capacity of container B: "))

target = int(input("Enter the required capacity: "))

def moves(state):

a , b = state

possible\_moves = []

#empty A

if (0,b) not in visited:

possible\_moves.append((0,b))

#empty B

if (a,0) not in visited:

possible\_moves.append((a,0))

#fill A

if (A,b) not in visited:

possible\_moves.append((A,b))

#fill B

if (a,B) not in visited:

possible\_moves.append((a,B))

#A to B

if a+b>=B:

if (a-(B-b),B) not in visited:

possible\_moves.append((a-(B-b),B))

else:

if (0,a+b) not in visited:

possible\_moves.append((0,a+b))

#B to A

if a+b>=A:

if (A,b-(A-a)) not in visited:

possible\_moves.append((A,b-(A-a)))

else:

if (a+b,0) not in visited:

possible\_moves.append((a+b,0))

return possible\_moves

def BFS():

start = (0,0)

open = [start]

visited.append(start)

found\_solution = False

path = []

while open:

current = open.pop(0)

traversal.append(current)

if target in current:

found\_solution = True

path = [current]

while current != start:

path.insert(1,current)

current = parent\_state[current]

path.insert(1,start)

print("solution found : " , end=" ")

for state in path:

print(state, end=" -> " if state != path[-1] else "\n")

continue

next\_moves = moves(current)

for move in next\_moves:

if move not in visited:

open.append(move)

visited.append(move)

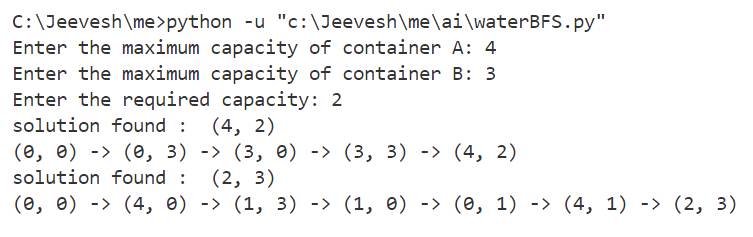
parent\_state[move] = current

if not found\_solution:

print("No solution")

BFS()

**Output:**



**b) Solution using DFS:**

**Algorithm:**



**Code:**

#water jug problem using DFS

A = int(input("Enter the maximum capacity of container A: "))

B = int(input("Enter the maximum capacity of container B: "))

target = int(input("Enter the required capacity: "))

visited = set()

def DFS(a , b , A , B , target , visited):

if a == target or b == target:

return [[(a,b)]]

visited.add((a,b))

next\_states = []

#empty A

if (0,b) not in visited:

next\_states.append((0,b))

#empty B

if (a,0) not in visited:

next\_states.append((a,0))

#fill A

if (A,b) not in visited:

next\_states.append((A,b))

#fill B

if (a,B) not in visited:

next\_states.append((a,B))

#A to B

if a+b>=B:

if (a-(B-b),B) not in visited:

next\_states.append((a-(B-b),B))

else:

if (0,a+b) not in visited:

next\_states.append((0,a+b))

#B to A

if a+b>=A:

if (A,b-(A-a)) not in visited:

next\_states.append((A,b-(A-a)))

else:

if (a+b,0) not in visited:

next\_states.append((a+b,0))

all\_paths = []

for state in next\_states:

paths = DFS(state[0],state[1],A,B,target,visited)

if paths:

for path in paths:

all\_paths.append([(a,b)] + path)

return all\_paths

paths = DFS(0,0,A,B,target,visited)

if paths:

print("All Solutions:")

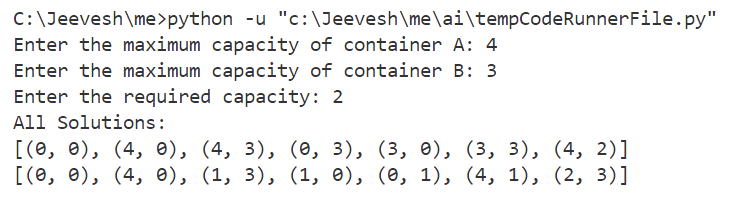
for path in paths:

print([step for step in path])

else:

print("No solution found")

**Output:**



**Conclusion**

The Water Jug problem has been solved by using the BFS and DFS approaches successfully.