3.Write a program to implement Water jug program using AI.

Source Code:

```
from collections import deque
def water_jug_problem(jug1_capacity, jug2_capacity, target):
    queue = deque([(0, 0, 0)])  # (jug1, jug2, steps)
    visited = set((0, 0))
    while queue:
        jug1, jug2, steps = queue.popleft()
        if jug1 == target or jug2 == target:
            return steps
        if (jug1_capacity, jug2) not in visited:
            queue.append((jug1_capacity, jug2, steps + 1))
            visited.add((jug1_capacity, jug2))
        if (jug1, jug2_capacity) not in visited:
            queue.append((jug1, jug2_capacity, steps + 1))
            visited.add((jug1, jug2_capacity))
        if (0, jug2) not in visited:
            queue.append((0, jug2, steps + 1))
            visited.add((0, jug2))
        if (jug1, 0) not in visited:
            queue.append((jug1, 0, steps + 1))
            visited.add((jug1, 0))
        pour_amount = min(jug1, jug2_capacity - jug2)
        if (jug1 - pour_amount, jug2 + pour_amount) not in
visited:
            queue.append((jug1 - pour_amount, jug2 + pour_amount,
steps + 1))
            visited.add((jug1 - pour_amount, jug2 + pour_amount))
        pour_amount = min(jug2, jug1_capacity - jug1)
        if (jug1 + pour_amount, jug2 - pour_amount) not in
visited:
            queue.append((jug1 + pour_amount, jug2 - pour_amount,
steps + 1))
            visited.add((jug1 + pour_amount, jug2 - pour_amount))
    return -1
jug1_capacity = 3
jug2_capacity = 5
target = 4
steps = water_jug_problem(jug1_capacity, jug2_capacity, target)
if steps != -1:
    print("Solution found in", steps, "steps.")
    print("Jug 1:", jug1_capacity)
    print("Jug 2:", jug2_capacity)
    print("Target:", target)
else:
    print("No solution found.")
```

OUTPUT

```
Solution found in 6 steps.
Jug 1: 3
Jug 2: 5
Target: 4
```

7. Build an Artificial Neural Network by implementing the Back propagation Algorithm and test the same using appropriate data sets

Source Code:

```
import numpy as np
X=np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y =np.array(([92], [86], [89]), dtype=float)
X=X/np.amax(X,axis=0)
def sigmoid (x):
    return (1/(1 + np.exp(-x)))
def derivatives_sigmoid(x):
    return x * (1- x)
epoch=7000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons)
)
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
 hinp1=np.dot(X,wh)
 hinp=hinp1 + bh
 hlayer_act =sigmoid(hinp)
 outinp1=np.dot(hlayer_act,wout)
 outinp= outinp1+ bout
 output = sigmoid(outinp)
 EO=y-output
 outgrad = derivatives_sigmoid(output)
 d_output = EO* outgrad
 EH=d_output.dot(wout.T)
 hiddengrad = derivatives_sigmoid(hlayer_act)
 d_hiddenlayer = EH * hiddengrad
 wout += hlayer_act.T.dot(d_output) *lr
 bout += np.sum(d_output, axis=0,keepdims=True) *lr
 wh +=X.T.dot(d_hiddenlayer) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

OUTPUT

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.999999  ]
 [0.99999749]
 [0.99999891]]
```

```
P_absent_and_friday = 0.03
P_friday = 0.20


P_absent_given_friday = P_absent_and_friday / P_friday


print(f'Probability that a student is absent given that today is
Friday: {P_absent_given_friday:.2f}')


print("\nDetailed Calculation:")
print(f"P(Absent and Friday) = {P_absent_and_friday:.2f}")
print(f"P(Friday) = {P_friday:.2f}")
print(f"P(Absent | Friday) = P(Absent and Friday) / P(Friday)")
print(f"P(Absent | Friday) = {P_absent_and_friday:.2f} /
{P_friday:.2f}")
print(f"P(Absent | Friday) = {P_absent_given_friday:.2f}")


print("\nVerification:")
expected_result = 0.03 / 0.20
print(f"Verification of result: {expected_result:.2f}")
assert P_absent_given_friday == expected_result, "The calculated
result does not match the expected result."

print("The calculation is correct and verified.")

OUTPUT Will Be

Detailed Calculation:
P(Absent and Friday) = 0.03
P(Friday) = 0.20
P(Absent | Friday) = P(Absent and Friday) / P(Friday)
P(Absent | Friday) = 0.03 / 0.20
P(Absent | Friday) = 0.15

Verification:
Verification of result: 0.15
The calculation is correct and verified.
```

a* algoritham

```
import heapq
def a_star_search(grid, start, goal):
    open_list = []
    heapq.heappush(open_list, (0, start))
    came_from = {start: None}
    cost_so_far = {start: 0}
    while open_list:
        _, current = heapq.heappop(open_list)
        if current == goal:
            break
        for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
            next = (current[0] + dx, current[1] + dy)
            if 0 <= next[0] < len(grid) and 0 <= next[1] <
len(grid[0]) and grid[next[0]][next[1]] != 1:
                new_cost = cost_so_far[current] + 1
                if next not in cost_so_far or new_cost <
cost_so_far[next]:
                    cost_so_far[next] = new_cost
                    priority = new_cost + abs(next[0] - goal[0]) +
abs(next[1] - goal[1])
                    heapq.heappush(open_list, (priority, next))
                    came_from[next] = current
    return came_from, cost_so_far
grid = [
    [0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 0, 1],
    [0, 1, 1, 0, 0],
    [0, 0, 0, 0, 0]
]
start = (0, 0)
goal = (4, 4)
came_from, cost_so_far = a_star_search(grid, start, goal)
current = goal
path = []
while current:
    path.append(current)
    current = came_from[current]
path.reverse()
print(path)
```

OUTPUT

```
[(0, 0), (0, 1), (1, 1), (2, 1), (2, 2), (2, 3), (3, 3), (3, 4),
(4, 4)]
```