Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 28-2-2021	Date of Submission: 28-2-2022

Write a program for water jug problem using Pruning Technique

Water Jug Problem is also known as Water Pouring Puzzles, measuring puzzles and decanting problems. These belong to a class of puzzles, in which there are a finite and specific number of water jugs having predefined integral capacities, in terms of gallons or litres.

The prime challenge in the water jug problem is that these water jugs do not have any calibrations for measuring the intermediate water levels. In order to solve these puzzles, you are given a required measurement that you have to achieve by transferring waters from one jug to another, you can iterate multiple times until the final goal is reached, but in order to get the best results, the final solution should have a minimum cost and a minimum number of transfers.

Problem Statement:

Given two water jugs with capacities **X** and **Y** liters. Initially, both the jugs are empty. Also given that there is an infinite amount of water available. The jugs do not have markings to measure smaller quantities.
One can perform the following operations on the jug:

- Fill any of the jugs completely with water.
- Pour water from one jug to the other until one of the jugs is either empty or full, (X, Y) -> (X – d, Y + d)
- Empty any of the jugs

The task is to determine whether it is possible to measure **Z** liters of water using both the jugs. And if true, print any of the possible ways.

Algorithm

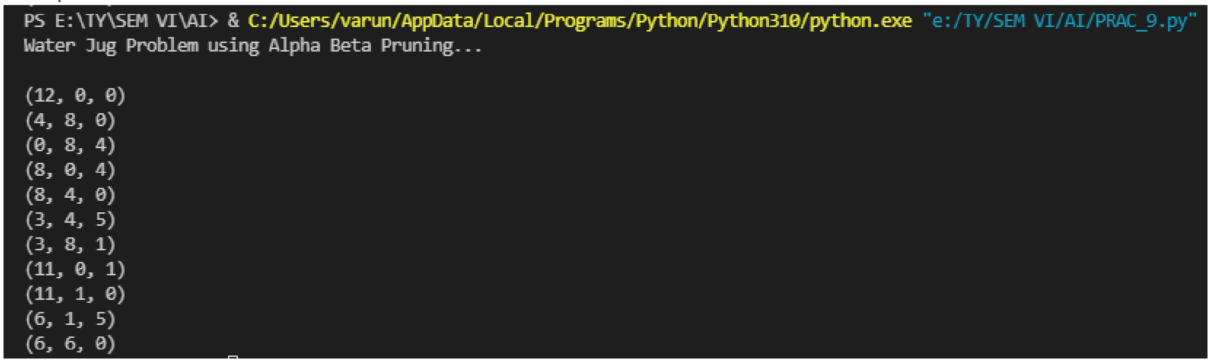
- Initialise a queue to implement BFS.
- Since, initially, both the jugs are empty, insert the state {0, 0} into the queue.
- Perform the following state, till the queue becomes empty:
  - Pop out the first element of the queue.
  - If the value of popped element is equal to Z, return True.
  - Let X_left and Y_left be the amount of water left in the jugs respectively.
  - Now perform the fill operation:
    - If the value of X_left < X, insert ({X_left, Y}) into the hashmap, since this state hasn’t been visited and some water can still be poured in the jug.
    - If the value of Y_left < Y, insert ({Y_left, X}) into the hashmap, since this state hasn’t been visited and some water can still be poured in the jug.
  - Perform the **empty** operation:
    - If the state ({0, Y_left}) isn’t visited, insert it into the hashmap, since we can empty any of the jugs.

    - Similarly, if the state ({X_left, 0) isn’t visited, insert it into the hashmap, since we can empty any of the jugs.
  - Perform the **transfer of water** operation:
    - min({X-X_left, Y}) can be poured from second jug to first jug. Therefore, in case – {X + min({X-X_left, Y}) , Y – min({X-X_left, Y}) isn’t visited, put it into hashmap.
    - min({X_left, Y-Y_left}) can be poured from first jug to second jug. Therefore, in case – {X_left – min({X_left, Y – X_left}) , Y + min({X_left, Y – Y_left}) isn’t visited, put it into hashmap.
- Return False, since, it is not possible to measure **Z** litres.

Code

```python
capacity = (12,8,5)
x = capacity[0]
y = capacity[1]
z = capacity[2]

memory = {}

ans = []

def get_all_states(state):
    a = state[0]
    b = state[1]
    c = state[2]

    if(a==6 and b==6):
        ans.append(state)
        return True

    if((a,b,c) in memory):
        return False

    memory[(a,b,c)] = 1

    if(a>0):
        if(a+b<=y):
            if( get_all_states((0,a+b,c)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((a-(y-b), y, c)) ):
                ans.append(state)
                return True
        if(a+c<=z):
            if( get_all_states((0,b,a+c)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((a-(z-c), b, z)) ):
                ans.append(state)
                return True

    if(b>0):
        if(a+b<=x):
            if( get_all_states((a+b, 0, c)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((x, b-(x-a), c)) ):
                ans.append(state)
                return True
        if(b+c<=z):
            if( get_all_states((a, 0, b+c)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((a, b-(z-c), z)) ):
                ans.append(state)
                return True

    if(c>0):
        if(a+c<=x):
            if( get_all_states((a+c, b, 0)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((x, b, c-(x-a))) ):
                ans.append(state)
                return True
        if(b+c<=y):
            if( get_all_states((a, b+c, 0)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((a, y, c-(y-b))) ):
                ans.append(state)
                return True

    return False

initial_state = (12,0,0)
print("Water Jug Problem using Alpha Beta Pruning...\n")
get_all_states(initial_state)
ans.reverse()
for i in ans:
    print(i)
```

Output



```
PS E:\TY\SEM VI\AI> & C:/Users/varun/AppData/Local/Programs/Python/Python310/python.exe "e:/TY/SEM VI/AI/PRAC_9.py"
Water Jug Problem using Alpha Beta Pruning...

(12, 0, 0)
(4, 8, 0)
(0, 8, 4)
(8, 0, 4)
(8, 4, 0)
(3, 4, 5)
(3, 8, 1)
(11, 0, 1)
(11, 1, 0)
(6, 1, 5)
(6, 6, 0)
```