PRACTICAL – 2

AIM : Write a python program to solve a Water Jug Problem by using Breadth first search (BFS).

Code:

```
import time

max_jug1 = int(input("Enter the capacity of Jug 1: "))
max_jug2 = int(input("Enter the capacity of Jug 2: "))

class Node:
    def __init__(self, x=0, y=0, parent=None):
        self.x = x
        self.y = y
        self.parent = parent

goal_x = int(input("Enter the goal for Jug 1: "))
goal_y = int(input("Enter the goal for Jug 2: "))

class BFSAlgorithm:
    def __init__(self):
        self.queue = []

    def bfs_search(self):
        start_node = Node(0, 0, None)
        self.queue.append(start_node)
        visited = set()

        while self.queue:
            current_node = self.queue.pop(0)

            if current_node.x == goal_x and current_node.y == goal_y:
                return current_node

            for new_node in self.gen_successors(current_node):
                state = (new_node.x, new_node.y)
                if state not in visited:
                    visited.add(state)
                    self.queue.append(new_node)

        return None
```

```
    def gen_successors(self, node):
        successors = []
        x, y = node.x, node.y

        if x < max_jug1:
            successors.append(Node(max_jug1, y, node))
        if y < max_jug2:
            successors.append(Node(x, max_jug2, node))
        if x > 0:
            successors.append(Node(0, y, node))
        if y > 0:
            successors.append(Node(x, 0, node))
        transfer = min(x, max_jug2 - y)
        if transfer > 0:
            successors.append(Node(x - transfer, y + transfer, node))
        transfer = min(y, max_jug1 - x)
        if transfer > 0:
            successors.append(Node(x + transfer, y - transfer, node))
        if x + y <= max_jug1 and y > 0:
            successors.append(Node(x + y, 0, node))
        if x + y <= max_jug2 and x > 0:
            successors.append(Node(0, x + y, node))
        return successors

start_time = time.time()
bfs_solver = BFSAlgorithm()
result_node = bfs_solver.bfs_search()
end_time = time.time()
execution_time = (end_time - start_time) * 1000

if result_node:
    path = []
    while result_node:
        path.append((result_node.x, result_node.y))
        result_node = result_node.parent
    path.reverse()
    print("\nSolution Path:")
    for step in path:
        print(step)
    print("Step Needed to Sove Problem :", len(path))
else:
    print("\nNo solution found!")
```

print(f"\nTime taken: {execution_time:.2f} ms")

Output:

```
Enter the capacity of Jug 1: 4
Enter the capacity of Jug 2: 3
Enter the goal for Jug 1: 2
Enter the goal for Jug 2: 0

Solution Path:
(0, 0)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
(0, 2)
(2, 0)
Step Needed to Sove Problem : 7

Time taken: 0.31 ms
```