1

```python
import collections

def bfs(graph, root):
    visited, queue = set(), collections.deque([root])
    visited.add(root)
    while queue:
        vertex = queue.popleft()
        print(str(vertex) + " ", end="")
        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end="")
    for next in graph[start]:
        print("->", end="")
        dfs(graph, next, visited)
    return visited

if __name__ == '__main__':
    graph = {0: [1, 2], 1: [], 2: [4, 5], 3: [5], 4: [], 5: []}
    print("Following is Breadth First Traversal: ")
    bfs(graph, 0)
    print("")
    print("Following is Depth First Traversal: ")
    dfs(graph, 0)
    print(" ")
```

```
ishadp@pop-os:~/Documents/Code/AI$ /bin/python3 /home/ishadp/Documents/Code/Practical-AI/1.py
Following is Breadth First Traversal:
0 1 2 4 5
Following is Depth First Traversal:
0->1->2->4->5
```

2

```python

from collections import deque

def water_jug_problem(capacity_a, capacity_b, target):
    queue = deque([((0, 0), [])])
    visited = set()

    while queue:
        (a, b), path = queue.popleft()
        if a == target or b == target:
            return path + [(a, b)]
        next_states = [
            (capacity_a, b),
            (a, capacity_b),
            (0, b),
            (a, 0),
            (min(a + b, capacity_a), max(0, a + b - capacity_a)),
            (max(0, a + b - capacity_b), min(a + b, capacity_b))
        ]

        for next_state in next_states:
            if next_state not in visited:
                visited.add(next_state)
                queue.append((next_state, path + [(a, b)]))

    return None

cap1 = 5
cap2 = 8
tgt = 4
path = water_jug_problem(cap1, cap2, tgt)
if path is None:
    print("No possible Solution")
else:
    for state in path:
        print(state)
cap1 = 4
cap2 = 3
tgt = 2
path = water_jug_problem(cap1, cap2, tgt)
if path is None:
    print(" No possible Solution")
else:
    for state in path:
        print(state)


```

```
ishadp@pop-os:~/Documents/Code/AI$ /bin/python3 /home/ishadp/Documents/Code/Practical-AI/2.py
(0, 0)
(5, 0)
(0, 5)
(5, 5)
(2, 8)
(2, 0)
(0, 2)
(5, 2)
(0, 7)
(5, 7)
(4, 8)
(0, 0)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
ishadp@pop-os:~/Documents/Code/AI$
```