EX.NO:01 **Problem Solving using State Space Search**

DATE:27.7.2022 **Uninformed Search Strategies**

AIM:

To solve the given Water Jug problem using BFS and DFS.

BFS:

Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.

BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.

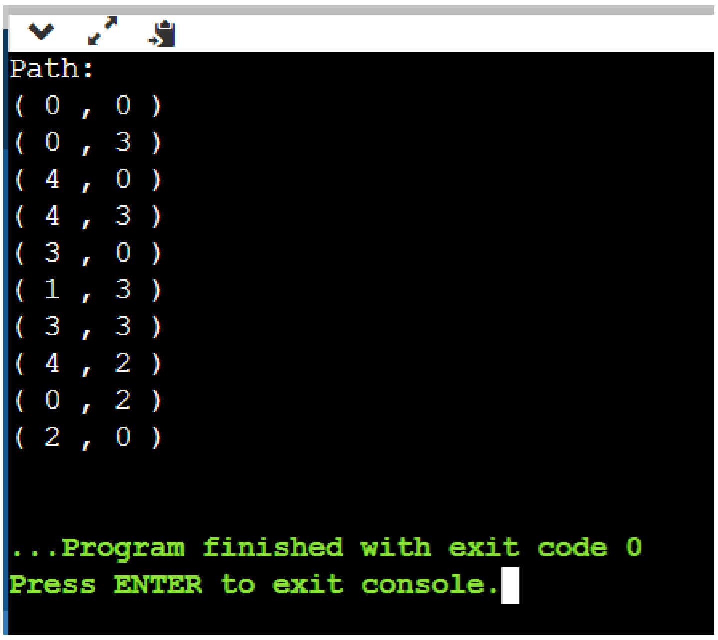
The breadth-first search algorithm is an example of a general-graph search algorithm.

Breadth-first search implemented using FIFO queue data structure.

SOURCE CODE:

```
from collections import deque

def BFS(a, b, target):

    m = {}

    flag = False

    path = []

    q = deque()

    q.append((0, 0))


    while (len(q) > 0):

        curr = q.popleft()

        if ((curr[0], curr[1]) in m):

            continue

        if ((curr[0] > a or curr[1] > b or curr[0] < 0 or curr[1] < 0)):

            continue

        path.append([curr[0], curr[1]])

        m[(curr[0], curr[1])] = 1

        if (curr[0] == target or curr[1] == target):
            flag = True
            if (curr[0] == target):
                if (curr[1] != 0):
                    path.append([curr[0], 0])
            else:
                if (curr[0] != 0):
                    path.append([0, curr[1]])
                    path.append([curr[1],0])
            l = len(path)
            for i in range(l):
                print("(", path[i][0], ",",path[i][1], ")")
            break
        q.append([curr[0], b])
        q.append([a, curr[1]])
        for f in range(max(a, b) + 1):
            c = curr[0] + f
            d = curr[1] - f
            if (c == a or (d == 0 and d >= 0)):
                q.append([c, d])

            c = curr[0] - f
            d = curr[1] + f
            if ((c == 0 and c >= 0) or d == b):
                q.append([c, d])
        q.append([a, 0])
        q.append([0, b])
    if (not flag):
        print("No solution")

if __name__ == '__main__':
    Jug1, Jug2, target = 4, 3, 2
    print("Path:")
    BFS(Jug1, Jug2, target)
```

OUTPUT:



```
Path:
( 0 , 0 )
( 0 , 3 )
( 4 , 0 )
( 4 , 3 )
( 3 , 0 )
( 1 , 3 )
( 3 , 3 )
( 4 , 2 )
( 0 , 2 )
( 2 , 0 )


...Program finished with exit code 0
Press ENTER to exit console.
```

DFS:

Depth-first search isa recursive algorithm for traversing a tree or graph data structure.

It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.

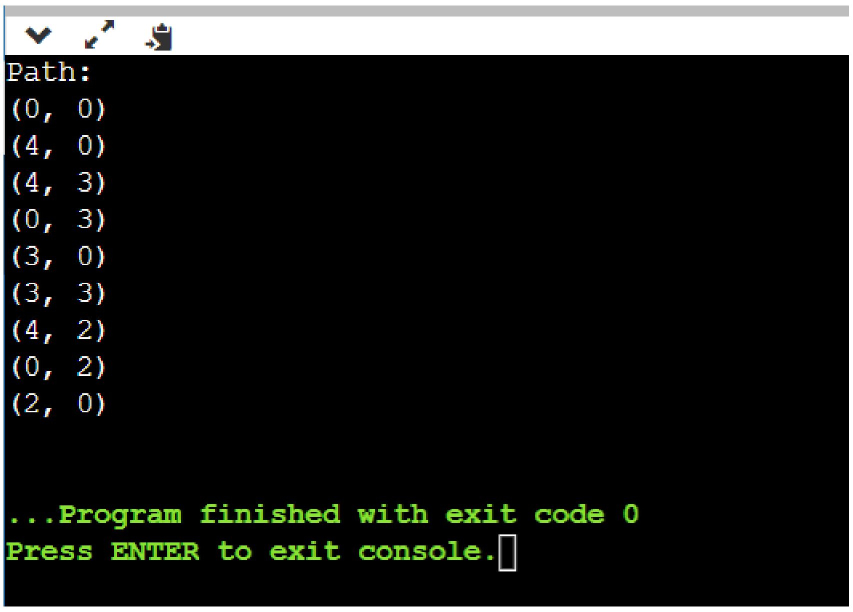
DFS uses a stack data structure for its implementation.

The process of the DFS algorithm is similar to the BFS algorithm.

SOURCE CODE:

```
from collections import defaultdict
visited = defaultdict(lambda: False)
J1, J2, L = 0, 0, 0
def DFS(X, Y):
    global J1, J2, L
    if (X == L and Y == 0) or (Y == L and X == 0):
        if(X == L):
            print("(",X, ", ",Y,")", sep ="")
        elif(Y==L):
            print("(",X, ", ",Y,")", sep ="")
            print("(",Y, ", ",X,")", sep ="")
        return True

    if visited[(X, Y)] == False:
        print("(",X, ", ",Y,")", sep ="")

        visited[(X, Y)] = True

        return (DFS(0, Y) or
                DFS(X, 0) or
                DFS(J1, Y) or
                DFS(X, J2) or
                DFS(X + min(Y, (J1-X)),
                Y - min(Y, (J1-X))) or
                DFS(X - min(X, (J2-Y)),
                Y + min(X, (J2-Y))))
    else:
        return False
J1 = 4
J2 = 3
L = 2
print("Path:")
DFS(0, 0)
```

OUTPUT:



```
Path:
(0, 0)
(4, 0)
(4, 3)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
(0, 2)
(2, 0)


...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

The given Water Jug Problem is solved using BFS and DFS successfully.