# AI (2180703)

## Tutorial 3

**Name : Vivek Visavadiya**
**Enrollment No. : 170200107124**
**Division/Batch : F/F3**

Que. Write a program to implement DFS ( for 8 puzzle problem or water jug Problem or any AI search problem.)

## Program(practical3.py):

```
import collections


def main():
    starting_node = [[0, 0]]
    jugs = get_jugs()
    goal_amount = get_goal(jugs)
    check_dict = {}
    is_depth = get_search_type()
    search(starting_node, jugs, goal_amount, check_dict, is_depth)

def get_index(node):
    return pow(7, node[0]) * pow(5, node[1])

def get_search_type():
    s =input("Enter 'b' for BFS, 'd' for DES: ")
    s = s[0].lower()

    while s != 'd' and    s!= 'b':
        s = input("The input is not valid! Enter 'b' for BFS, 'd' for DFS: ")
        s = s[0].lower()
    return s == 'd'

def get_jugs():
```

```
    print(“Receiving the volume of the jugs…”)
    jugs = []

    temp = int(input(“Enter first jug volume (>1): “))
    while temp < 1:
        temp = int(input(“Enter a valid amount (>1): “))
    jugs.append(temp)

    temp = int(input(“Enter second jug volume (>1): “))
    while temp < 1:
        temp = int(input(“Enter a valid amount (>1): “))
    jugs.append(temp)

    return jugs

def get_goal(jugs):
    print(“Receiving the desired amount of the water…”)

    max_amount = max(jugs[0], jugs[1])
    s = "Enter the desired amount of water (1 - {0}): ".format(max_amount)
    goal_amount = int(input(s))
    while goal_amount < 1 or goal_amount > max_amount:
        goal_amount = int(input("Enter a valid amount (1 - {0}): ".form
        at(max_amount)))
    return goal_amount

def is_goal(path, goal_amount):
    print("Checking if the goal is achieved...")
    return path[-1][0] == goal_amount or path[-1][1] == goal_amount

def been_there(node, check_dict):
    print("Checking if {0} is visited before...".format(node))
    return check_dict.get(get_index(node), False)

def next_transitions(jugs, path, check_dict):
    print("Finding next transitions and checking for the loops...")
```

```
result = []
next_nodes = []
 node = []
 a_max = jugs[0]
 b_max = jugs[1]

a = path[-1][0]
b = path[-1][1]

node.append(a_max)
 node.append(b)
if not been_there(node, check_dict):
        next_nodes.append(node)
node = []

node.append(a)
node.append(b_max)
if not been_there(node, check_dict):
        next_nodes.append(node)
node = []

node.append(min(a_max, a + b))
 node.append(b - (node[0] - a)) # b - ( a' - a)
 if not been_there(node, check_dict):
        next_nodes.append(node)
node = []

node.append(min(a + b, b_max))
 node.insert(0, a - (node[0] - b))
if not been_there(node, check_dict):
         next_nodes.append(node)
 node = []

node.append(0)
 node.append(b)
if not been_there(node, check_dict):
         next_nodes.append(node)
```

```
        node = []

        node.append(a)
        node.append(0)
        if not been_there(node, check_dict):
                next_nodes.append(node)
        node = []

        for i in range(0, len(next_nodes)):
                temp = list(path)
                temp.append(next_nodes[i])
                result.append(temp)

        if len(next_nodes) == 0:
                print("No more unvisited nodes...\nBacktracking...")
        else:
                print("Possible transitions: ")
                for nnode in next_nodes:
                        print(nnode)

                return result

def transition(old, new, jugs):
        a = old[0]
        b = old[1]
        a_prime = new[0]
        b_prime = new[1]
        a_max = jugs[0]
        b_max = jugs[1]

        if a > a_prime:
                if b == b_prime:
                        return "Clear {0}-liter jug:\t\t\t".format(a_max)
                else:
                        return "Pour {0}-liter jug into {1}-liter jug:\t".format(a_ max,
                b_max)
```

```
        else:
                if b > b_prime:
                        if a == a_prime:
                                return "Clear {0}-liter jug:\t\t\t".format(b_max)
                        else:
                                return "Pour {0}-liter jug into {1}-liter jug:\t".forma
                        t(b_max, a_max)
                else:
                        if a == a_prime:
                                return "Fill {0}-liter jug:\t\t\t".format(b_max)
                        else:
                                return "Fill {0}-liter jug:\t\t\t".format(a_max)

def print_path(path, jugs):

        print("Starting from:\t\t\t\t", path[0])
         for i in range(0, len(path) - 1):
                 print(i+1,":", transition(path[i], path[i+1], jugs), path[i+1])

def search(starting_node, jugs, goal_amount, check_dict, is_depth):

        if is_depth:
                 print("Implementing DFS...")
        else:
                print("Implementing BFS...")
        goal = []
        accomplished = False
        q = collections.deque()
        q.appendleft(starting_node)

        while len(q) != 0:
                path = q.popleft()
                check_dict[get_index(path[-1])] = True
                if len(path) >= 2:
                        print(transition(path[-2], path[-1], jugs), path[-1])
                if is_goal(path, goal_amount):
                        accomplished = True
```

```
                goal = path
                break

            next_moves = next_transitions(jugs, path, check_dict)
            for i in next_moves:
                if is_depth:
                    q.appendleft(i)
                else:
                    q.append(i)

        if accomplished:
            print("The goal is achieved\nPrinting the sequence of the moves.
        ..\n")
            print_path(goal, jugs)
        else:
            print("Problem cannot be solved.")

if __name__ == '__main__':
        main()
```

**OUTPUT:**

C:\Windows\System32\cmd.exe

```
G:\SEM 8\AI>python practical3.py
Receiving the volume of the jugs...
Enter first jug volume (>1): 4
Enter second jug volume (>1): 3
Receiving the desired amount of the water...
Enter the desired amount of water (1 - 4): 2
Enter 'b' for BFS, 'd' for DFS: d
Implementing DFS...
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [4, 0] is visited before...
Checking if [0, 3] is visited before...
Checking if [0, 0] is visited before...
Checking if [0, 0] is visited before...
Checking if [0, 0] is visited before...
Checking if [0, 0] is visited before...
Possible transitions:
[4, 0]
[0, 3]
Fill 3-liter jug:                          [0, 3]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [4, 3] is visited before...
Checking if [0, 3] is visited before...
Checking if [3, 0] is visited before...
Checking if [0, 3] is visited before...
Checking if [0, 3] is visited before...
Checking if [0, 0] is visited before...
Possible transitions:
[4, 3]
[3, 0]
Pour 3-liter jug into 4-liter jug:         [3, 0]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [4, 0] is visited before...
Checking if [3, 3] is visited before...
Checking if [3, 0] is visited before...
Checking if [0, 3] is visited before...
Checking if [0, 0] is visited before...
Checking if [3, 0] is visited before...
Possible transitions:
[4, 0]
[3, 3]
Fill 3-liter jug:                          [3, 3]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [4, 3] is visited before...
Checking if [3, 3] is visited before...
Checking if [4, 2] is visited before...
Checking if [3, 3] is visited before...
Checking if [0, 3] is visited before...
Checking if [3, 0] is visited before...
Possible transitions:
[4, 3]
[4, 2]
Pour 3-liter jug into 4-liter jug:         [4, 2]
Checking if the gaol is achieved...
The goal is achieved
Printing the sequence of the moves...

Starting from:                             [0, 0]
1 : Fill 3-liter jug:                      [0, 3]
2 : Pour 3-liter jug into 4-liter jug:     [3, 0]
3 : Fill 3-liter jug:                      [3, 3]
4 : Pour 3-liter jug into 4-liter jug:     [4, 2]

G:\SEM 8\AI>
```