AI(2180703)

Tutorial - 3

Name : harshil khandhar
Enrollment No. : 170200107053
Division/Batch : E/E4

Q. Write a program to implement DFS (for 8 puzzle problem or Water Jug problem or any AI search problem).

Code (DFS.py):

```
#Write a program to implement DFS (for 8 puzzle
#problem or Water Jug problem or any AI search
#problem)

import collections

def main():

    starting_node = [[0, 0]]
    jugs = get_jugs()
    goal_amount = get_goal(jugs)
    check_dict = {}
    search(starting_node, jugs, goal_amount, check_dict,True)

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

def get_jugs():

    jugs = []

    jugA = int(input("\nEnter 1st Jug Volume(>1): "))
    jugs.append(jugA)

    jugB = int(input("\nEnter 2nd Jug Volume(>1): "))
    jugs.append(jugB)

    return jugs

def get_goal(jugs):

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

    return goal_amount

def is_goal(path, goal_amount):
    return path[-1][0] == goal_amount or path[-1][1] == goal_amount
```

```
def been_there(node, check_dict):
    return check_dict.get(get_index(node), False)

def next_transitions(jugs, path, check_dict):

    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))
    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)

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

    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".format(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("\nStarting 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("\nImplementing DFS.....")

    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:
            pass
        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:
        s = "\nThe goal is achieved\n\n"
```

```
        t = "Printing the sequence of the moves..."
        print(s+t)
        print_path(goal, jugs)
    else:
        print("\nProblem cannot be solved.")


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

Output :

```
C:\Users\hp\Desktop\SEM-8\AI\PRACTICAL>py DFS.py

Enter 1st Jug Volume(>1): 4

Enter 2nd Jug Volume(>1): 3

Enter the desired amount of water (1 - 4): 2

Implementing DFS.....

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]
```