CSE 401: Artificial Intelligence

Water Jug Problem using BFS DFS

Utkarsh Gupta
A2305217557
7CSE 8Y

August 07th, 2020

1 Water Jug Problem using BFS & DFS

Given Problem: You are given a m liter jug and a n liter jug where $0 < m < n$. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. You have to use the jugs to measure d liters of water where $d < n$. Determine the minimum no of operations to be performed to obtain d liters of water in one of jug.

States: Amount of water in each respective jug, where the states are represented by [S(J1), S(J2)] and S(J1) is the amount in the first jug and S(J2) is the amount in the second jug.

Capacity of Jug1 (J1): 3 litres.

Capacity of Jug1 (J2): 4 litres.

Initial State: [0, 0]

Goal State: The user give the input the amount of water required in the jug (J2) i.e. [2, y] or [2, 0].

These are the initial operators used:

Operators:
1. Fill the first jug.
2. Fill the second jug.
3. Empty the first jug.
4. Empty the second jug.
5. Pour the first jug into the second jug.
6. Pour the second jug into the second jug.

Branching Factor: 6 (because we have 6 operators)

2 Code

[1]:

```
import collections
```

[2]:

```
#This method return a key value for a given node.
#Node is a list of two integers representing current state of the jugs
def get_index(node):
    return pow(7, node[0]) * pow(5, node[1])
```

[3]:

```
#This method accepts an input for asking the choice for type of searching␣
 ↪required i.e. BFS or DFS.
#Method return True for BFS, False otherwise
def get_search_type():
    s = input("Enter 'b' for BFS, 'd' for DFS: ")
    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'
```

[4]:

```
#This method accept volumes of the jugs as an input from the user.
#Returns a list of two integeres representing volumes of the jugs.
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
```

[5]:

```
#This method accepts the desired amount of water as an input from the user␣
 ↪whereas
#the parameter jugs is a list of two integers representing volumes of the jugs
#Returns the desired amount of water as goal
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}): ".
 ↪format(max_amount)))

    return goal_amount
```

[6]:

```
#This method checks whether the given path matches the goal node.
#The path parameter is a list of nodes representing the path to be checked
#The goal_amount parameter is an integer representing the desired amount of water
def is_goal(path, goal_amount):

    print("Checking if the gaol is achieved...")

    return path[-1][0] == goal_amount or path[-1][1] == goal_amount
```

[7]:

```
#This method validates whether the given node is already visited.
#The parameter node is a list of two integers representing current state of the␣
 ↪jugs
#The parameter check_dict is   a dictionary storing visited nodes
def been_there(node, check_dict):

    print("Checking if {0} is visited before...".format(node))

    return check_dict.get(get_index(node), False)
```

[8]:

```
#This method returns the list of all possible transitions
#The parameter jugs is a list of two integers representing volumes of the jugs
#The parameter path is a list of nodes represeting the current path
#The parameter check_dict is a dictionary storing visited nodes
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]  # initial amount in the first jug
    b = path[-1][1]  # initial amount in the second jug
    # 1. fill in the first jug
    node.append(a_max)
    node.append(b)
    if not been_there(node, check_dict):
        next_nodes.append(node)
    node = []

    # 2. fill in the second jug
    node.append(a)
    node.append(b_max)
    if not been_there(node, check_dict):
        next_nodes.append(node)
    node = []

    # 3. second jug to first jug
    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 = []

    # 4. first jug to second jug
    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 = []

    # 5. empty first jug
    node.append(0)
    node.append(b)
    if not been_there(node, check_dict):
        next_nodes.append(node)
    node = []

    # 6. empty second jug
    node.append(a)
    node.append(0)
    if not been_there(node, check_dict):
        next_nodes.append(node)

    # create a list of next paths
    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
```

[9]:

```
# This method returns a string explaining the transition from old state/node to␣
 ↪new state/node
# The parameter old is a list representing old state/node
# The parameter new is a list representing new state/node
# The parameter jugs is a list of two integers representing volumes of the jugs

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)
```

[10]:

```
#This method prints the goal path
#The path is a list of nodes representing the goal path
#The jugs is a list of two integers representing volumes of the jugs
```

```
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])
```

[11]:

```
#This method searches for a path between starting node and goal node
# The parameter starting_node is a list of list of two integers representing␣
 ↪initial state of the jugs
#The parameter jugs a list of two integers representing volumes of the jugs
#The parameter goal_amount is an integer represting the desired amount
#The parameter check_dict is  a dictionary storing visited nodes
#The parameter is_breadth is implements BFS, if True; DFS otherwise
def search(starting_node, jugs, goal_amount, check_dict, is_breadth):

    if is_breadth:
        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_breadth:
                q.append(i)
            else:
                q.appendleft(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.")
```

[12]:
```
if __name__ == '__main__':
    starting_node = [[0, 0]]
    jugs = get_jugs()
    goal_amount = get_goal(jugs)
    check_dict = {}
    is_breadth = get_search_type()
    search(starting_node, jugs, goal_amount, check_dict, is_breadth)
```

```
Receiving the volume of the jugs...
Enter first jug volume (>1): 3
Enter second jug volume (>1): 7
Receiving the desired amount of the water...
Enter the desired amount of water (1 - 7): 5
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 [3, 0] is visited before...
Checking if [0, 7] 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:
[3, 0]
[0, 7]
Fill 3-liter jug:                        [3, 0]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 0] is visited before...
Checking if [3, 7] 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:
[3, 7]
[0, 3]
Fill 7-liter jug:                        [0, 7]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 7] is visited before...
Checking if [0, 7] is visited before...
Checking if [3, 4] is visited before...
Checking if [0, 7] is visited before...
Checking if [0, 7] is visited before...
Checking if [0, 0] is visited before...
Possible transitions:
[3, 7]
[3, 4]
Fill 7-liter jug:                        [3, 7]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 7] is visited before...
Checking if [3, 7] is visited before...
Checking if [3, 7] is visited before...
Checking if [3, 7] is visited before...
Checking if [0, 7] is visited before...
Checking if [3, 0] is visited before...
No more unvisited nodes...
Backtracking...
Pour 3-liter jug into 7-liter jug:       [0, 3]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 3] is visited before...
Checking if [0, 7] 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:
[3, 3]
Fill 3-liter jug:                        [3, 7]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 7] is visited before...
Checking if [3, 7] is visited before...
Checking if [3, 7] is visited before...
Checking if [3, 7] is visited before...
Checking if [0, 7] is visited before...
Checking if [3, 0] is visited before...
No more unvisited nodes...
Backtracking...
Pour 7-liter jug into 3-liter jug:       [3, 4]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 4] is visited before...
Checking if [3, 7] is visited before...
Checking if [3, 4] is visited before...
Checking if [0, 7] is visited before...
Checking if [0, 4] is visited before...
Checking if [3, 0] is visited before...
Possible transitions:
```

```
[0, 4]
Fill 3-liter jug:                        [3, 3]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 3] is visited before...
Checking if [3, 7] is visited before...
Checking if [3, 3] is visited before...
Checking if [0, 6] is visited before...
Checking if [0, 3] is visited before...
Checking if [3, 0] is visited before...
Possible transitions:
[0, 6]
Clear 3-liter jug:                       [0, 4]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 4] is visited before...
Checking if [0, 7] is visited before...
Checking if [3, 1] is visited before...
Checking if [0, 4] is visited before...
Checking if [0, 4] is visited before...
Checking if [0, 0] is visited before...
Possible transitions:
[3, 1]
Pour 3-liter jug into 7-liter jug:       [0, 6]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 6] is visited before...
Checking if [0, 7] is visited before...
Checking if [3, 3] is visited before...
Checking if [0, 6] is visited before...
Checking if [0, 6] is visited before...
Checking if [0, 0] is visited before...
Possible transitions:
[3, 6]
Pour 7-liter jug into 3-liter jug:       [3, 1]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 1] is visited before...
Checking if [3, 7] is visited before...
Checking if [3, 1] is visited before...
Checking if [0, 4] is visited before...
Checking if [0, 1] is visited before...
Checking if [3, 0] is visited before...
Possible transitions:
[0, 1]
Fill 3-liter jug:                        [3, 6]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
```

```
Checking if [3, 6] is visited before...
Checking if [3, 7] is visited before...
Checking if [3, 6] is visited before...
Checking if [2, 7] is visited before...
Checking if [0, 6] is visited before...
Checking if [3, 0] is visited before...
Possible transitions:
[2, 7]
Clear 3-liter jug:                       [0, 1]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 1] is visited before...
Checking if [0, 7] is visited before...
Checking if [1, 0] is visited before...
Checking if [0, 1] is visited before...
Checking if [0, 1] is visited before...
Checking if [0, 0] is visited before...
Possible transitions:
[1, 0]
Pour 3-liter jug into 7-liter jug:       [2, 7]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 7] is visited before...
Checking if [2, 7] is visited before...
Checking if [3, 6] is visited before...
Checking if [2, 7] is visited before...
Checking if [0, 7] is visited before...
Checking if [2, 0] is visited before...
Possible transitions:
[2, 0]
Pour 7-liter jug into 3-liter jug:       [1, 0]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 0] is visited before...
Checking if [1, 7] is visited before...
Checking if [1, 0] is visited before...
Checking if [0, 1] is visited before...
Checking if [0, 0] is visited before...
Checking if [1, 0] is visited before...
Possible transitions:
[1, 7]
Clear 7-liter jug:                       [2, 0]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 0] is visited before...
Checking if [2, 7] is visited before...
Checking if [2, 0] is visited before...
Checking if [0, 2] is visited before...
Checking if [0, 0] is visited before...
Checking if [2, 0] is visited before...
Possible transitions:
[0, 2]
Fill 7-liter jug:                        [1, 7]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 7] is visited before...
Checking if [1, 7] is visited before...
Checking if [3, 5] is visited before...
Checking if [1, 7] is visited before...
Checking if [0, 7] is visited before...
Checking if [1, 0] is visited before...
Possible transitions:
[3, 5]
Pour 3-liter jug into 7-liter jug:       [0, 2]
Checking if the gaol is achieved...
Finding next transitions and checking for the loops...
Checking if [3, 2] is visited before...
Checking if [0, 7] is visited before...
Checking if [2, 0] is visited before...
Checking if [0, 2] is visited before...
Checking if [0, 2] is visited before...
Checking if [0, 0] is visited before...
Possible transitions:
[3, 2]
Pour 7-liter jug into 3-liter jug:       [3, 5]
Checking if the gaol is achieved...
The goal is achieved
Printing the sequence of the moves...

Starting from:                           [0, 0]
1 : Fill 7-liter jug:                    [0, 7]
2 : Pour 7-liter jug into 3-liter jug:   [3, 4]
3 : Clear 3-liter jug:                   [0, 4]
4 : Pour 7-liter jug into 3-liter jug:   [3, 1]
5 : Clear 3-liter jug:                   [0, 1]
6 : Pour 7-liter jug into 3-liter jug:   [1, 0]
7 : Fill 7-liter jug:                    [1, 7]
8 : Pour 7-liter jug into 3-liter jug:   [3, 5]
```