
Exercise 3 – State Space Search: Decantation Problem (Water Jug Problem)

Date: **01/09/2022**

Aim:

Statement: You are given two jugs, a 4-litre one and a 3-litre one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 litres of water into the 4-litre jug.

To perform the following:

1. Formulate the problem: Identify state, initial state, goal state, conditions, actions and state space tree.
2. Use a suitable data structure to keep track of the parent of every state. Write a function to print all possible solution sequences from the initial state to the goal state (number of solutions).
3. Write a function next state (S) that returns a list of successor states of a given state 'S'.
4. Implement the following Search Algorithms to search the state space tree for a goal state that produces the required sequence of pouring's from the initial state and its path cost.
   (a) BFS
   (b) DFS
   (c) DLS with limit=6
   (d) IDS

Code:

```
#State space search - Decantation problem (Water Jug problem)

import copy

class jar(object):
    __slots__ = ['cur', 'max']

    def __init__(self, cur, max):
        self.cur = cur
        self.max = max

    def __eq__(self, other):
        if self.cur == other.cur and self.max == other.max:
            return True
```

```
        return False

    def __ne__(self, other):
        return not self == other

    def __str__(self):
        return str(self.cur)

    def __hash__(self):
        return hash((self.cur, self.max))

    def isFull(self):
        return self.cur == self.max

    def spaceLeft(self):
        return self.max - self.cur


class state(object):
    __slots__ = ['jars', 'parent']

    def __init__(self, jars, parent=None):
        self.jars = jars
        self.parent = parent

    def __eq__(self, other):
        flag = True
        for i in range(2):
            if self.jars[i] != other.jars[i]:
                flag = False
                break

        return flag

    def __str__(self):
        s = "("
        for jar in self.jars:
            s += str(jar) + ', '
        s = s[:-2]
        s += ')'
        return s

    def __hash__(self):
```

```
        return hash(self.jars)


initial_state = state((jar(0, 4), jar(0, 3)))
goal_state1 = state((jar(2, 4), jar(3, 3)))
goal_state2 = state((jar(2, 4), jar(0, 3)))

def next_states(curr_state):
    next = []
    for i in range(len(curr_state.jars)):
        for j in range(len(curr_state.jars)):
            if i == j:
                continue
            jars_temp = copy.deepcopy(curr_state.jars)

            #No capacity left to transfer from jar 'i'
            if jars_temp[i].cur <= 0:
                jars_temp[i].cur = jars_temp[i].max
                next.append(state(jars_temp, parent=curr_state))
                continue
            #pour to other jar
            if jars_temp[j].spaceLeft() > 0:
                avail = jars_temp[j].spaceLeft()

                if jars_temp[i].cur < avail:
                    jars_temp[j].cur += jars_temp[i].cur
                    jars_temp[i].cur = 0
                    next.append(state(jars_temp, parent=curr_state))
                    continue

                else:
                    jars_temp[j].cur += avail
                    jars_temp[i].cur -= avail
                    next.append(state(jars_temp, parent=curr_state))
                    continue
            #drain
            draintemp = jars_temp[i].cur
            jars_temp[i].cur = 0
            next.append(state(jars_temp, parent=curr_state))
            jars_temp[i].cur = draintemp
            jars_temp[j].cur = 0
            next.append(state(jars_temp, parent=curr_state))
    return next
```

```
print("_____")

def print_steps(current_state):

    #Base case, this is the root state
    if current_state.parent is None:
        print(current_state)
        return

    #Recursive step
    print_steps(current_state.parent)
    print(current_state)

b1 = []

def bfs(initial_state, goal_state, trace=False):
    print("Tracing the queue contents: ")
    visited = set()

    states = []
    states.append(initial_state)
    b1.append(initial_state)

    while len(states) != 0:
        if trace:
            for s in states:
                print(s, end=' ')
            print()

        current_state = states.pop(0)
        if current_state == goal_state:
            print("Found Goal State!\n")
            #print_steps(current_state)
            break
        successors = next_states(current_state)

        b1.append(initial_state)
        for new_state in successors:
            if new_state not in visited:
                visited.add(new_state)
                states.append(new_state)
```

```
            b1.append(new_state)
        else:
            #Duplicate state
            pass


def dfs(initial_state, goal_state, trace=False):
    visited = set()

    states = []
    states.append(initial_state)

    while len(states) != 0:
        current_state = states.pop(0)
        if current_state == goal_state:

            print_steps(current_state)
            print("Found Goal State!")
            break
        successors = next_states(current_state)

        for new_state in successors:
            if new_state not in visited:
                visited.add(new_state)
                states.append(new_state)
            else:
                #Duplicate state
                pass

def dls(initial_state, goal_state, limit, trace=False):
    visited = set()
    states = []
    states.append(initial_state)
    count = 0

    while ((len(states) != 0) and count < limit):
        current_state = states.pop(0)
        if current_state == goal_state:
            print_steps(current_state)
            print("Found Goal State!")
            break
        successors = next_states(current_state)
        count += 1
```

```
        for new_state in successors:
            if new_state not in visited:
                visited.add(new_state)
                states.append(new_state)
            else:
                #Duplicate state
                pass

def ids(L,gs):
  print(initial_state)
  for m in L:
    if m == initial_state:
      continue
    print(m)
    if m == gs:
      print("Goal state found!\n")
      break

#trace = True for queue contents
print("BFS: ")
bfs(initial_state, goal_state1, trace=True)
print("_______")
b2 = []
b2 = b1
b1 = []
bfs(initial_state, goal_state2, trace=True)
print("_______")

print("DFS: ")
dfs(initial_state, goal_state1, trace=True)
print("_______")
dfs(initial_state, goal_state2, trace=True)
print("_______")

print("\nDLS: ")
dls(initial_state, goal_state1, limit = 6, trace=True)
print("_______")
dls(initial_state, goal_state2, limit = 6, trace=True)
print("_______")

print("\nIDS: ")
ids(b1,goal_state1)
```

```
print("_______")
ids(b1,goal_state2)
print("_______")
```

Output:

```
~/AIwork$ python3 jugStates.py
______
BFS:
Tracing the queue contents:
(0, 0)
(4, 0) (0, 3)
(0, 3) (1, 3) (4, 3)
(1, 3) (4, 3) (3, 0)
(4, 3) (3, 0) (1, 0)
(3, 0) (1, 0)
(1, 0) (3, 3)
(3, 3) (0, 1)
(0, 1) (4, 2)
(4, 2) (4, 1)
(4, 1) (0, 2)
(0, 2) (2, 3)
(2, 3) (2, 0)
Found Goal State!

_______
Tracing the queue contents:
(0, 0)
(4, 0) (0, 3)
(0, 3) (1, 3) (4, 3)
(1, 3) (4, 3) (3, 0)
(4, 3) (3, 0) (1, 0)
(3, 0) (1, 0)
(1, 0) (3, 3)
(3, 3) (0, 1)
(0, 1) (4, 2)
(4, 2) (4, 1)
(4, 1) (0, 2)
(0, 2) (2, 3)
(2, 3) (2, 0)
(2, 0)
Found Goal State!
```

```
_______
DFS:
(0, 0)
(4, 0)
(1, 3)
(1, 0)
(0, 1)
(4, 1)
(2, 3)
Found Goal State!
_______
(0, 0)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
(0, 2)
(2, 0)
Found Goal State!
_______
```

```
DLS:
(0, 0)
(4, 0)
(0, 3)
(1, 3)
(4, 3)
(3, 0)
(1, 0)
(3, 3)
(0, 1)
(4, 2)
(4, 1)
(0, 2)
(2, 3)
Goal state found!

_______
(0, 0)
(4, 0)
(0, 3)
(1, 3)
(4, 3)
(3, 0)
(1, 0)
(3, 3)
(0, 1)
(4, 2)
```

```
(4, 1)
(0, 2)
(2, 3)
(2, 0)
Goal state found!


_______

IDS:
(0, 0)
(4, 0)
(0, 3)
(1, 3)
(4, 3)
(3, 0)
(1, 0)
(3, 3)
(0, 1)
(4, 2)
(4, 1)
(0, 2)
(2, 3)
Goal state found!
```

```
_______
(0, 0)
(4, 0)
(0, 3)
(1, 3)
(4, 3)
(3, 0)
(1, 0)
(3, 3)
(0, 1)
(4, 2)
(4, 1)
(0, 2)
(2, 3)
(2, 0)
Goal state found!


_______
~/AIwork$
```