SSN College of Engineering

Department of Computer Science and Engineering

CS1504 — Artificial Intelligence

2021 – 2022

Assignment — 02
(State Space Search — Decantation Problem)

July 23, 2021

Problem Statement

You are given an 8-litre jar full of water and two empty jars of 5- and 3-litre capacity. You have to get exactly 4 litres of water in one of the jars. You can completely empty a jar into another jar with space or completely fill up a jar from another jar.

1. Formulate the problem: Identify states, actions, initial state, goal state(s). Represent the state by a 3-tuple. For example, the initial state state is (8,0,0). (4,1,3) is a goal state (there may be other goal states also).
2. Use a suitable data structure to keep track of the parent of every state. Write a function to print the sequence of states and actions from the initial state to the goal state.
3. Write a function next states(s) that returns a list of successor states of a given state s.
4. Implement Breadth-First-Search algorithm to search the state space graph for a goal state that produces the required sequence of pouring's. Use a Queue as a frontier that stores the discovered states yet to be explored. Use a dictionary for exploration that is used to store the explored states.
5. Modify your program to trace the contents of the Queue in your algorithm. How many states are explored by your algorithm?

Responses

1. Formulated below (State Space Formulation)
2. A queue data structure is used to perform Breadth-First-Search (Queue.py)
   A dictionary with first explored parent of each state is maintained
3. Function implemented as *get_next_state(state)* (StateFunctions.py)
4. BFS implemented as function *BFS()*
5. The algorithm explores 16 distinct states

(*Link to **repl.it** implementation repository*)

State Space Formulation

<u>3-tuple Representation</u>

(a, b, c) where,

a is the amount of water in the 8-liter jar

b is the amount of water in the 5-liter jar

c is the amount of water in the 3-liter jar

<u>Initial State</u> : (8, 0, 0)

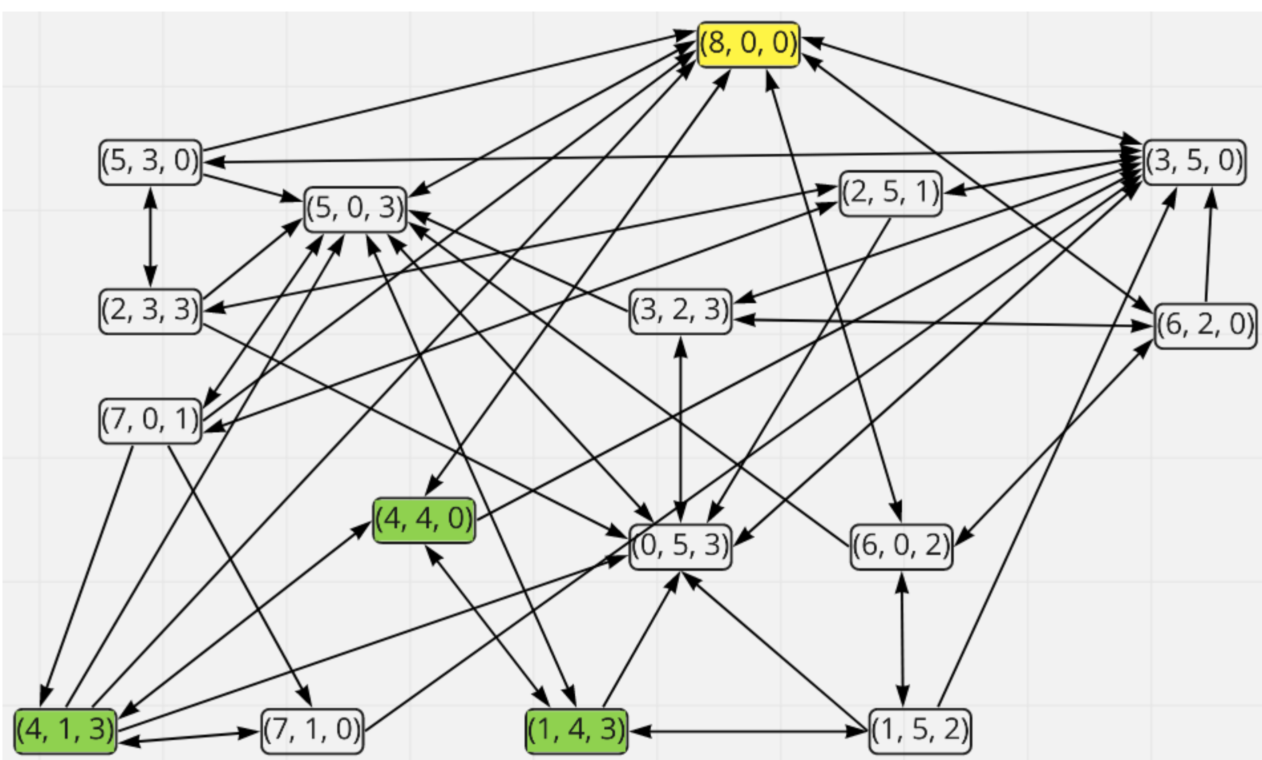
<u>Allowed actions to progress to Next State</u> :

1. Empty a jar into another with empty space
2. Fill up a jar completely from another jar

<u>Constraint</u>: Sum of quantity of water over all jars is always equal to 8 liters

<u>Goal State</u> : One of the three jars contains exactly 4L of water

State Transition Diagram



Python Program Code

1. Queue.py - Script for Queue ADT

```
class Queue:
   # [HEAD, ....... , TAIL]
   def __init__(self, data_list=None):
       self.data = list()
       self.size = 0
       if data_list is not None:
           self.data.extend(data_list)
           self.size = len(data_list)

   def enqueue(self, data_list):
       self.data.extend(data_list)
       self.size += len(data_list)

   def dequeue(self):
       if self.size == 0:
           return None
       self.size -= 1
       return self.data.pop(0)

   def is_empty(self):
       return self.size==0
```

2. StateFunctions.py - Functions to generate and evaluate states

```
# CONSTANTS
CAPACITY = (8,5,3)
NUM_JUGS = 3
INITIAL_STATE = (8,0,0)

def get_next_states(state):

   # Transfer water from jug 'from_' to 'to'
   def transfer(from_, to):
```

```
        # Find bottleneck
        space = CAPACITY[to] - state[to]
        water = state[from_]
        transit = min(space, water)
        # Perform transfer
        new_state = list(state)
        new_state[to] += transit
        new_state[from_] -= transit
        return tuple(new_state)

    # From i to j
    # Check all possible combinations
    result = list()
    for i in range(NUM_JUGS):
        for j in range(NUM_JUGS):
            if (i==j):
                # Self-transfer is meaningless
                continue
            if CAPACITY[j]==state[j]:
                # Cannot fill more
                # Destination full
                continue
            if state[i]==0:
                # Nothing to transfer
                # Source empty
                continue
            result.append(transfer(i,j))
    return result

def is_goal_state(state):
    return 4 in state
```

3. main.py - Driver function for BFS

```
from Queue import *
from StateFunctions import *
```

```python
line = "----------------------------"

def deduce_path(state, parents):

   def deduce_path_rec(state, path_seq):
       this_parent = parents[state]
       # Reached the root node
       if this_parent is None:
           return path_seq
       # Prepend node
       path_seq = [this_parent] + path_seq[:]
       return deduce_path_rec(this_parent, path_seq)

   return deduce_path_rec(state, [])

def BFS():

   state_space = Queue([INITIAL_STATE])
   goal_states = list()
   explored_states = set()
   parents = {INITIAL_STATE: None}

   while not state_space.is_empty():
       # Get next state and its path
       state = state_space.dequeue()
       # Skip iteration if state already explored
       if state in explored_states:
           continue
       explored_states.add(state)
       # Check if the state is a goal-state
       if is_goal_state(state):
           goal_states.append(state)
       # Find the next states
       fringe = get_next_states(state)
       state_space.enqueue(fringe)
       # Add parent of each fringe state, if not already added
       for new_state in fringe:
           if new_state not in parents:
```

```python
                parents[new_state] = state

    return goal_states, explored_states, parents

if __name__ == '__main__':

    goal_states, explored_states, parents = BFS()
    print(parents)

    print("\nDISTINCT EXPLORED STATES COUNT: ", len(explored_states))

    print("\nGOAL STATES COUNT: ", len(goal_states))

    print("\nINITIAL STATE")
    print(INITIAL_STATE)

    print("\nGOAL STATES")
    for state in goal_states:
        print(state)

    print("\nEXPLORED STATES")
    for state in explored_states:
        print(state)

    for state in goal_states:
        print("\nPATH to reach", state)
        print("\n".join(map(str, deduce_path(state, parents))))
        print(state, '--> GOAL STATE')
```

(Output on next page)

Output

```
DISTINCT EXPLORED STATES COUNT:  16

GOAL STATES COUNT:  3

INITIAL STATE
(8, 0, 0)

GOAL STATES
(1, 4, 3)
(4, 4, 0)
(4, 1, 3)

EXPLORED STATES
(8, 0, 0)
(3, 5, 0)
(5, 0, 3)
(0, 5, 3)
(3, 2, 3)
(5, 3, 0)
(6, 2, 0)
(2, 3, 3)
(6, 0, 2)
(2, 5, 1)
(1, 5, 2)
(7, 0, 1)
(1, 4, 3)
(7, 1, 0)
(4, 4, 0)
(4, 1, 3)
```

```
PATH 1
To reach (1, 4, 3)
----------------------------
(8, 0, 0)
(3, 5, 0)
(3, 2, 3)
(6, 2, 0)
(6, 0, 2)
(1, 5, 2)
(1, 4, 3) --> GOAL STATE
----------------------------

PATH 2
To reach (4, 4, 0)
----------------------------
(8, 0, 0)
(3, 5, 0)
(3, 2, 3)
(6, 2, 0)
(6, 0, 2)
(1, 5, 2)
(1, 4, 3)
(4, 4, 0) --> GOAL STATE
----------------------------

PATH 3
To reach (4, 1, 3)
----------------------------
(8, 0, 0)
(5, 0, 3)
(5, 3, 0)
(2, 3, 3)
(2, 5, 1)
(7, 0, 1)
(7, 1, 0)
(4, 1, 3) --> GOAL STATE
----------------------------
```