**SSN College of Engineering**

**Department of Computer Science and Engineering**

**CS1504—Artificial Intelligence**

**2020 – 2021**

**State Space Search—Decantation Problem**

**Session :** 2 **Name :** Rahul Ram M

**Date :** 24/08/2020 **Reg No :** 185001121

**Problem Formulation:**

**States:**

* States are represented by 3-tuple.
* a + b + c = 8
* 0 <= a <= 8
* 0 <= b <= 5
* 0 <= c <= 3

**Examples :** (6, 1, 1), (3, 4, 1)

**Actions:**

* Completely empty a jar into another jar with space.

(3, 2, 3) -> (6, 2, 0)

* Completely fill up a jar from another jar.

(4, 3, 1) -> (2, 5, 1)

**Initial State:**

(8, 0, 0)

**Goal State(s):**

Any one of the jars with exactly 4 litres of water.

**Examples :** (4, 1, 3), (1, 4, 3), (4, 4, 0)

**CODE:**

"""

jar1 - litres of water in jar1, jar2 - litres of water in jar2

Pour all water from the jar1 to jar2 till jar1 becomes empty or

jar2 becomes full.

returns the new value of jar1 and jar2.

"""

def fillOrEmpty(jar1, jar2, max):

if(max - jar2) < jar1:

return jar1 - (max-jar2), max

return 0, jar2+jar1

"""

Check if the new\_state is already present in the s\_list.

If not append the new\_state to the list.

"""

def addToList(new\_state):

for state in s\_list:

if state == new\_state:

return

queue.append(new\_state)

s\_list.append(new\_state)

"""

Takes a list of successors and print if the goal state is present.

Goal state - jar1 or jar2 should have value 4.

"""

def isFinalState(successors):

for state in successors:

if state[0] == 4 or state[1] == 4:

print("Initial State : (8, 0, 0)")

print("Final State :", state)

return True

return False

"""

Takes a list of state and prints it.

First tuple in the list is the parent and remaining states are

its successors.

"""

def printStates(successors):

first = True

for state in successors:

if(first):

print(state)

first = False

else:

print(" |->", state)

print()

"""

Takes a state as its input and returns the list of its successors

along with the same state.

"""

def findNextState(state):

max = (8, 5, 3)

successors =[state]

for i in range(3):

if state[i] != 0:

next1 = (i+1) % 3

next2 = (i+2) % 3

if(state[next1] != max[next1]):

new\_state = list(state)

new\_state[i], new\_state[next1] = fillOrEmpty(state[i], state[next1], max[next1])

successors.append(tuple(new\_state))

if(new\_state[i] != 0):

new\_state[i], new\_state[next2] = fillOrEmpty(new\_state[i], new\_state[next2], max[next2])

successors.append(tuple(new\_state))

if(state[next2] != max[next2]):

new\_state = list(state)

new\_state[i], new\_state[next2] = fillOrEmpty(state[i], state[next2], max[next2])

successors.append(tuple(new\_state))

if(new\_state[i] != 0):

new\_state[i], new\_state[next1] = fillOrEmpty(new\_state[i], new\_state[next1], max[next1])

successors.append(tuple(new\_state))

return successors

def main():

init\_state = (8, 0, 0)

# s\_list - list of all discovered states.

global s\_list

s\_list = [init\_state]

# queue - list to perform queue operation in BFS search.

global queue

queue = []

print("BFS Search......\n")

""" BFS part

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"""

successors = findNextState(init\_state)

for state in successors:

addToList(state)

printStates(successors)

while(isFinalState(successors) == False):

successors = findNextState(queue.pop(0))

for state in successors:

addToList(state)

printStates(successors)

"""\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"""

print("Number of states explored : ",len(s\_list) - len(queue))

main()

**Sample I/O:**

BFS Search......

(8, 0, 0)

|-> (3, 5, 0)

|-> (0, 5, 3)

|-> (5, 0, 3)

|-> (0, 5, 3)

(3, 5, 0)

|-> (0, 5, 3)

|-> (3, 2, 3)

|-> (5, 0, 3)

|-> (8, 0, 0)

(0, 5, 3)

|-> (5, 0, 3)

|-> (3, 5, 0)

(5, 0, 3)

|-> (0, 5, 3)

|-> (8, 0, 0)

|-> (5, 3, 0)

(3, 2, 3)

|-> (0, 5, 3)

|-> (5, 0, 3)

|-> (6, 2, 0)

|-> (3, 5, 0)

(5, 3, 0)

|-> (3, 5, 0)

|-> (0, 5, 3)

|-> (2, 3, 3)

|-> (0, 5, 3)

|-> (5, 0, 3)

|-> (8, 0, 0)

(6, 2, 0)

|-> (3, 5, 0)

|-> (0, 5, 3)

|-> (3, 2, 3)

|-> (0, 5, 3)

|-> (6, 0, 2)

|-> (8, 0, 0)

(2, 3, 3)

|-> (0, 5, 3)

|-> (5, 0, 3)

|-> (5, 3, 0)

|-> (2, 5, 1)

|-> (3, 5, 0)

(6, 0, 2)

|-> (1, 5, 2)

|-> (0, 5, 3)

|-> (5, 0, 3)

|-> (0, 5, 3)

|-> (8, 0, 0)

|-> (6, 2, 0)

(2, 5, 1)

|-> (0, 5, 3)

|-> (2, 3, 3)

|-> (5, 0, 3)

|-> (7, 0, 1)

|-> (3, 5, 0)

(1, 5, 2)

|-> (0, 5, 3)

|-> (1, 4, 3)

|-> (5, 0, 3)

|-> (6, 0, 2)

|-> (3, 5, 0)

Initial State : (8, 0, 0)

Final State : (1, 4, 3)

Number of states explored : 11