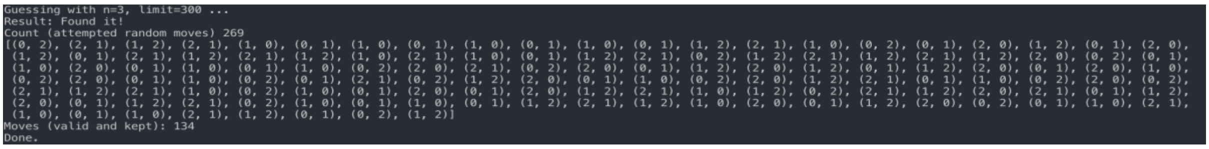
tower_of_hanoi.py (all 5 steps done)

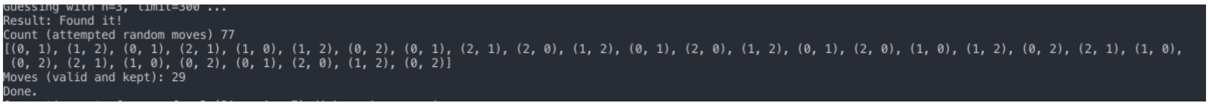
Tower of Hanoi is a puzzle created on Python. At step 2, a hard coding was done to path out the moves needed in order to complete the sequence. For step 3, the limit of 100 was not enough so it was changed to 300. The random search got upgraded in step 4. It reduced the length of the sequence, plus the chance of correct random guesses was also increased. Step 5 implements recursion, with the 'move' function called itself.

Before



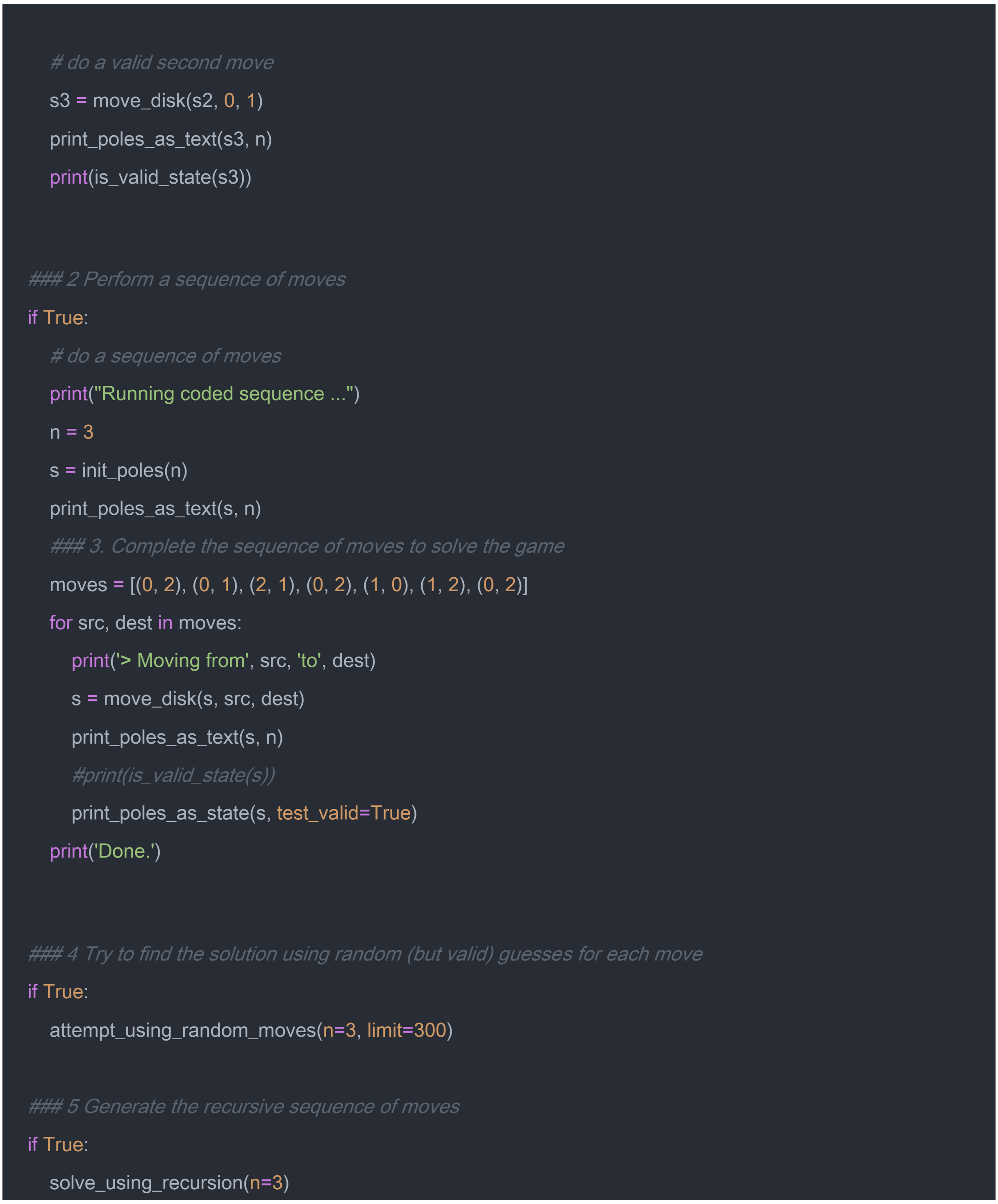
```
Guessing with n=3, limit=300 ...
Result: Found it!
Count (attempted random moves) 269
[(0, 2), (2, 1), (1, 2), (2, 1), (1, 0), (0, 1), (1, 0), (0, 1), (1, 0), (0, 1), (1, 0), (0, 1), (1, 2), (2, 1), (1, 0), (0, 2), (0, 1), (2, 0), (1, 2), (0, 1), (2, 0),
 (1, 2), (0, 1), (2, 1), (1, 2), (2, 1), (1, 2), (1, 0), (2, 1), (1, 0), (0, 1), (1, 2), (2, 1), (0, 2), (1, 2), (2, 1), (1, 2), (2, 1), (1, 2), (2, 0), (0, 2), (0, 1),
 (1, 0), (2, 0), (0, 1), (1, 0), (0, 1), (1, 0), (0, 2), (2, 0), (2, 1), (0, 2), (2, 0), (0, 1), (1, 2), (2, 0), (1, 2), (0, 1), (1, 2), (2, 0), (0, 1), (2, 0), (1, 0),
 (0, 2), (2, 0), (0, 1), (1, 0), (0, 2), (0, 1), (2, 1), (0, 2), (1, 2), (2, 0), (0, 1), (1, 0), (0, 2), (2, 0), (1, 2), (2, 1), (0, 1), (1, 0), (0, 2), (2, 0), (0, 2),
 (2, 1), (1, 2), (2, 1), (1, 0), (0, 2), (1, 0), (0, 1), (2, 0), (0, 1), (2, 0), (1, 2), (2, 1), (1, 0), (1, 2), (0, 2), (2, 1), (1, 2), (2, 0), (2, 1), (0, 1), (1, 2),
 (2, 0), (0, 1), (1, 2), (2, 1), (0, 2), (1, 0), (0, 1), (1, 0), (0, 1), (1, 2), (2, 1), (1, 2), (1, 0), (2, 0), (0, 1), (1, 2), (2, 0), (0, 2), (0, 1), (1, 0), (2, 1),
 (1, 0), (0, 1), (1, 0), (2, 1), (1, 2), (0, 1), (0, 2), (1, 2)]
Moves (valid and kept): 134
Done.
```

After



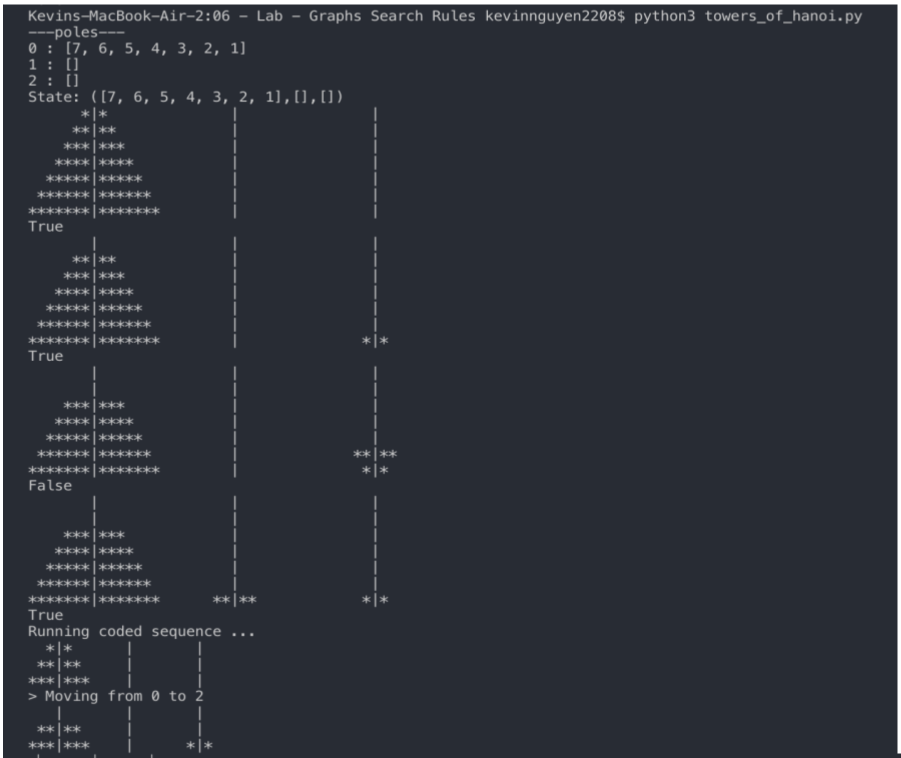
```
Guessing with n=3, limit=300 ...
Result: Found it!
Count (attempted random moves) 77
[(0, 1), (1, 2), (0, 1), (2, 1), (1, 0), (1, 2), (0, 2), (0, 1), (2, 1), (2, 0), (1, 2), (0, 1), (2, 0), (1, 2), (0, 1), (2, 0), (1, 0), (1, 2), (0, 2), (2, 1), (1, 0),
 (0, 2), (2, 1), (1, 0), (0, 2), (0, 1), (2, 0), (1, 2), (0, 2)]
Moves (valid and kept): 29
Done.
```

```
if __name__ == "__main__":

    ### 1 Simple test section.
    if True:
        # setup
        n = 7
        s1 = init_poles(n)
        # try each type of print
        print_poles(s1)
        print_poles_as_state(s1)
        print_poles_as_text(s1, n)
        # test for valid first state
        print(is_valid_state(s1))

        # move first disk
        s2 = move_disk(s1, 0, 2)
        print_poles_as_text(s2, n)
        print(is_valid_state(s2))

        # do an invalid move
        s3 = move_disk(s2, 0, 2)
        print_poles_as_text(s3, n)
        print(is_valid_state(s3))
```

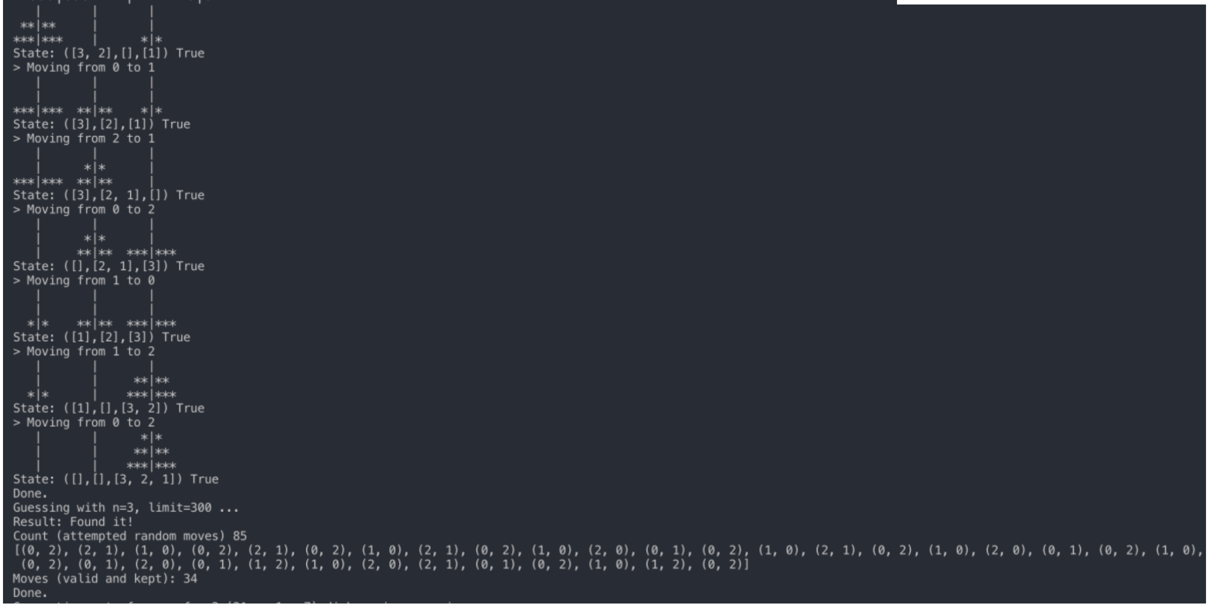


```
    # do a valid second move
    s3 = move_disk(s2, 0, 1)
    print_poles_as_text(s3, n)
    print(is_valid_state(s3))


### 2 Perform a sequence of moves
if True:
    # do a sequence of moves
    print("Running coded sequence ...")
    n = 3
    s = init_poles(n)
    print_poles_as_text(s, n)
    ### 3. Complete the sequence of moves to solve the game
    moves = [(0, 2), (0, 1), (2, 1), (0, 2), (1, 0), (1, 2), (0, 2)]
    for src, dest in moves:
        print('> Moving from', src, 'to', dest)
        s = move_disk(s, src, dest)
        print_poles_as_text(s, n)
        #print(is_valid_state(s))
        print_poles_as_state(s, test_valid=True)
    print('Done.')


### 4 Try to find the solution using random (but valid) guesses for each move
if True:
    attempt_using_random_moves(n=3, limit=300)

### 5 Generate the recursive sequence of moves
if True:
    solve_using_recursion(n=3)
```

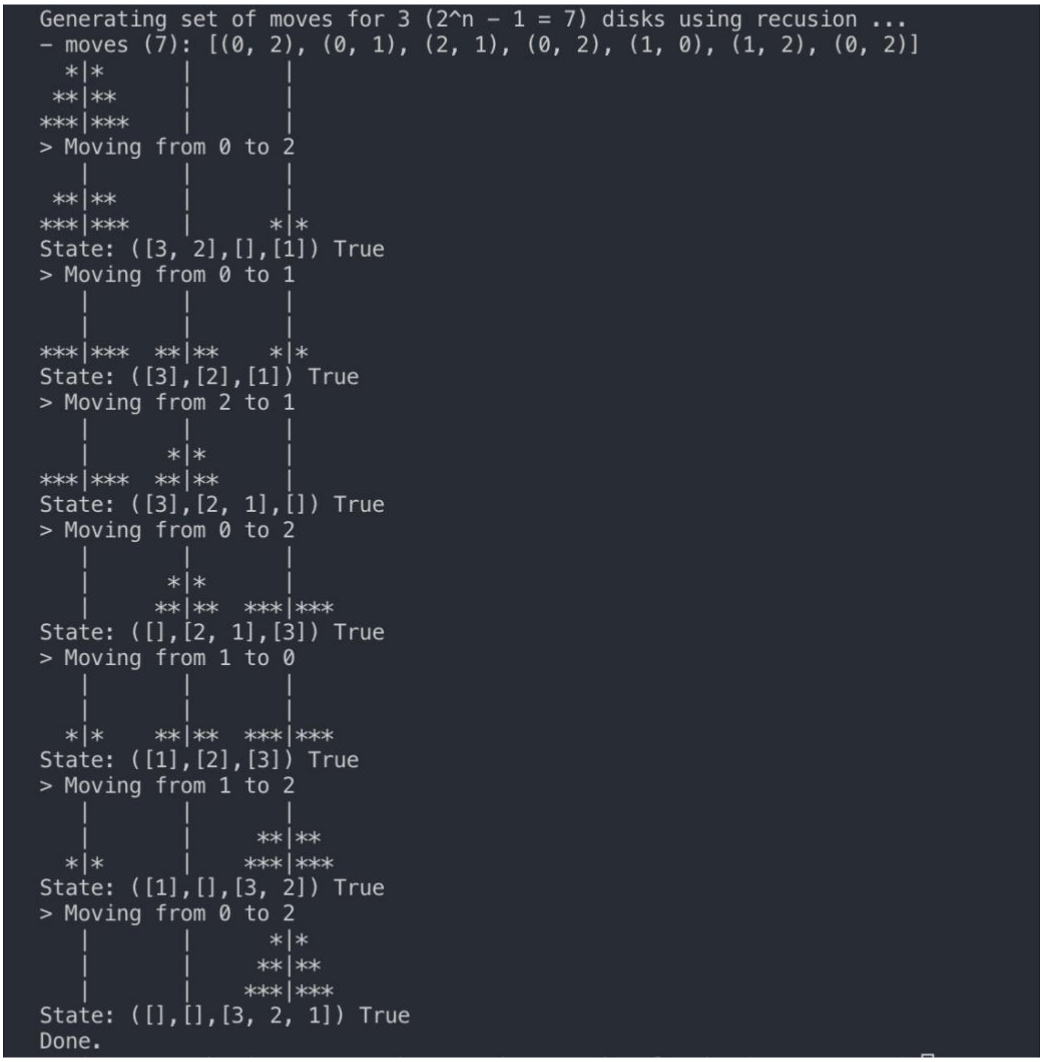
OUTPUT



```
Kevins-MacBook-Air-2:06 - Lab - Graphs Search Rules kevinnguyen2208$ python3 towers_of_hanoi.py
---poles---
0 : [7, 6, 5, 4, 3, 2, 1]
1 : []
2 : []
State: ([7, 6, 5, 4, 3, 2, 1],[],[])
      *|*              |               |
     **|**             |               |
    ***|***            |               |
   ****|****           |               |
  *****|*****          |               |
 ******|******         |               |
*******|*******        |               |
True
       |               |               |
     **|**             |               |
    ***|***            |               |
   ****|****           |               |
  *****|*****          |               |
 ******|******         |               |
*******|*******        |              *|*
True
       |               |               |
       |               |               |
    ***|***            |               |
   ****|****           |               |
  *****|*****          |               |
 ******|******         |             **|**
*******|*******        |              *|*
False
       |               |               |
       |               |               |
    ***|***            |               |
   ****|****           |               |
  *****|*****          |               |
 ******|******         |               |
*******|*******      **|**            *|*
True
Running coded sequence ...
  *|*       |       |
 **|**      |       |
***|***     |       |
> Moving from 0 to 2
   |        |       |
 **|**      |       |
***|***     |      *|*
```



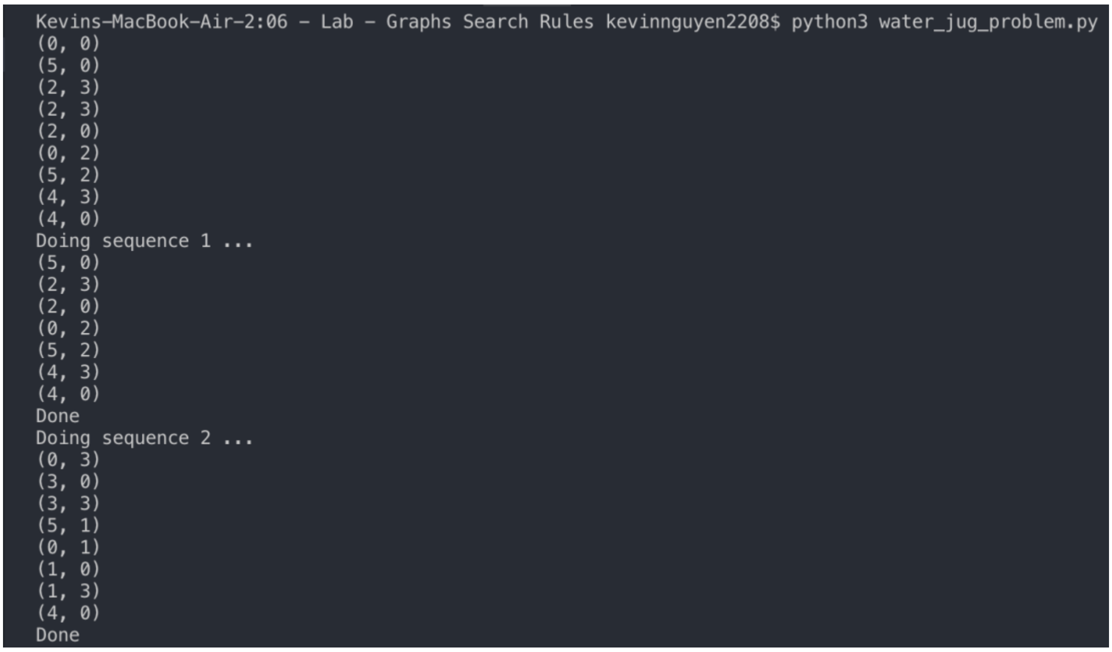
```
   |       |       |
 **|**     |       |
***|***    |      *|*
State: ([3, 2],[],[1]) True
> Moving from 0 to 1
   |       |       |
   |       |       |
***|***  **|**    *|*
State: ([3],[2],[1]) True
> Moving from 2 to 1
   |       |       |
   |      *|*      |
***|***  **|**     |
State: ([3],[2, 1],[]) True
> Moving from 0 to 2
   |       |       |
   |      *|*      |
   |     **|**  ***|***
State: ([],[2, 1],[3]) True
> Moving from 1 to 0
   |       |       |
   |       |       |
  *|*    **|**  ***|***
State: ([1],[2],[3]) True
> Moving from 1 to 2
   |       |       |
   |       |     **|**
  *|*      |    ***|***
State: ([1],[],[3, 2]) True
> Moving from 0 to 2
   |       |      *|*
   |       |     **|**
   |       |    ***|***
State: ([],[],[3, 2, 1]) True
Done.
Guessing with n=3, limit=300 ...
Result: Found it!
Count (attempted random moves) 85
[(0, 2), (2, 1), (1, 0), (0, 2), (2, 1), (0, 2), (1, 0), (2, 1), (0, 2), (1, 0), (2, 0), (0, 1), (0, 2), (1, 0), (2, 1), (0, 2), (1, 0), (2, 0), (0, 1), (0, 2), (1, 0),
 (0, 2), (0, 1), (2, 0), (0, 1), (1, 2), (1, 0), (2, 0), (2, 1), (0, 1), (0, 2), (1, 0), (1, 2), (0, 2)]
Moves (valid and kept): 34
Done.
```



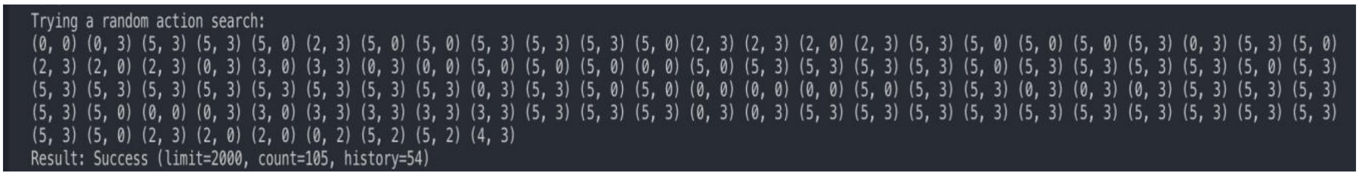
```
Generating set of moves for 3 (2^n - 1 = 7) disks using recusion ...
- moves (7): [(0, 2), (0, 1), (2, 1), (0, 2), (1, 0), (1, 2), (0, 2)]
  *|*      |       |
 **|**     |       |
***|***    |       |
> Moving from 0 to 2
   |       |       |
 **|**     |       |
***|***    |      *|*
State: ([3, 2],[],[1]) True
> Moving from 0 to 1
   |       |       |
   |       |       |
***|***  **|**    *|*
State: ([3],[2],[1]) True
> Moving from 2 to 1
   |       |       |
   |      *|*      |
***|***  **|**     |
State: ([3],[2, 1],[]) True
> Moving from 0 to 2
   |       |       |
   |      *|*      |
   |     **|**  ***|***
State: ([],[2, 1],[3]) True
> Moving from 1 to 0
   |       |       |
   |       |       |
  *|*    **|**  ***|***
State: ([1],[2],[3]) True
> Moving from 1 to 2
   |       |       |
   |       |     **|**
  *|*      |    ***|***
State: ([1],[],[3, 2]) True
> Moving from 0 to 2
   |       |      *|*
   |       |     **|**
   |       |    ***|***
State: ([],[],[3, 2, 1]) True
Done.
```

water_jug_problem.py (all 5 steps done)

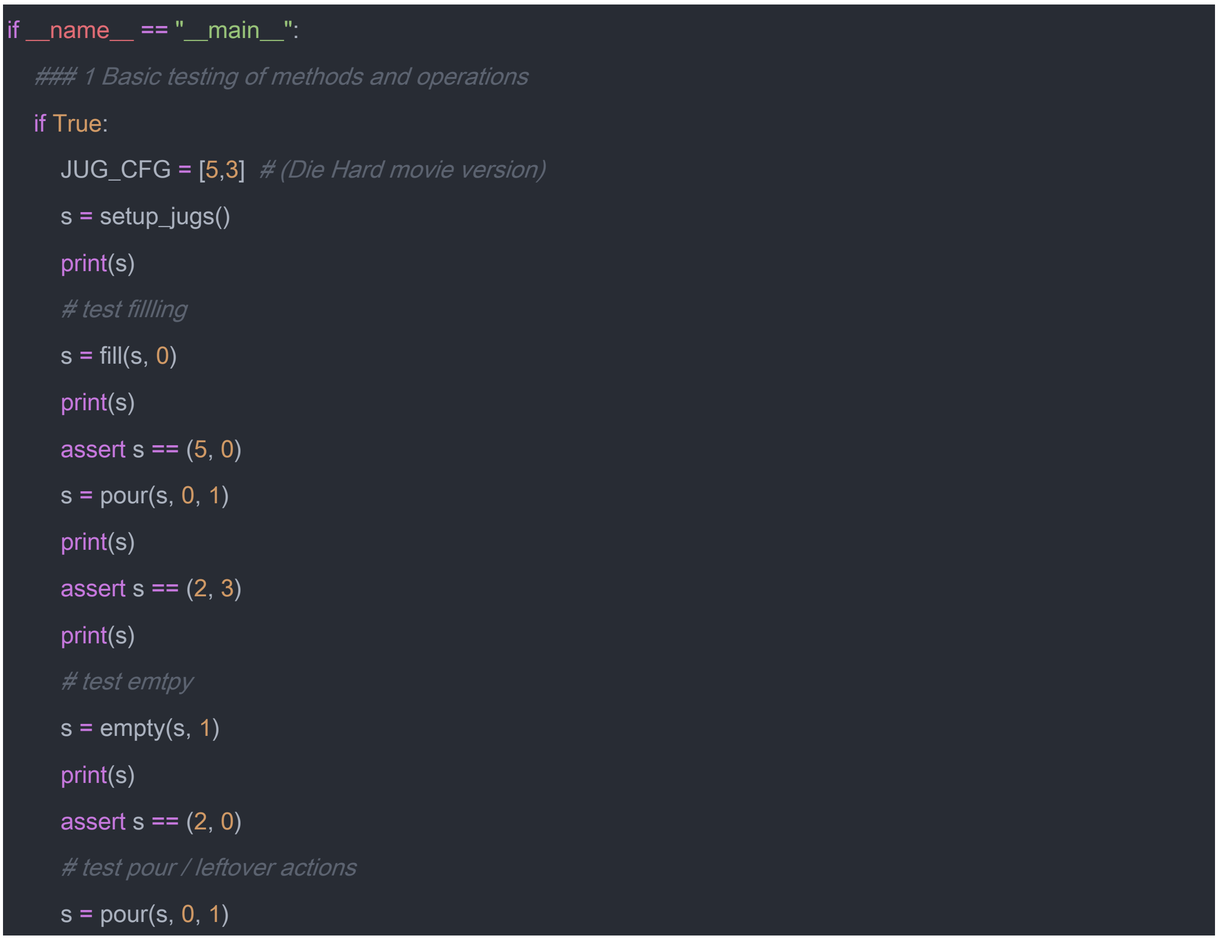
The water jug is a puzzle created on python. For step 6, asserts were fixed so that the moves worked successfully. In step 7, sequence 1 & 2 were solved successfully. At step 8, the limit was set to 4000. At step 9, random search got an upgrade in comparison to step 8, which gave out more frequent successful results. The solution path got smaller, and the guess count decreased.



```
Kevins-MacBook-Air-2:06 - Lab - Graphs Search Rules kevinnguyen2208$ python3 water_jug_problem.py
(0, 0)
(5, 0)
(2, 3)
(2, 3)
(2, 0)
(0, 2)
(5, 2)
(4, 3)
(4, 0)
Doing sequence 1 ...
(5, 0)
(2, 3)
(2, 0)
(0, 2)
(5, 2)
(4, 3)
(4, 0)
Done
Doing sequence 2 ...
(0, 3)
(3, 0)
(3, 3)
(5, 1)
(0, 1)
(1, 0)
(1, 3)
(4, 0)
Done
```



```
Trying a random action search:
(0, 0) (0, 3) (5, 3) (5, 3) (5, 0) (2, 3) (5, 0) (5, 0) (5, 3) (5, 3) (5, 3) (5, 0) (2, 3) (2, 3) (2, 0) (2, 3) (5, 3) (5, 0) (5, 0) (5, 0) (5, 3) (0, 3) (5, 3) (5, 0)
(2, 3) (2, 0) (2, 3) (0, 3) (3, 0) (3, 3) (0, 3) (0, 0) (5, 0) (5, 0) (5, 0) (0, 0) (5, 0) (5, 3) (5, 3) (5, 3) (5, 3) (5, 0) (5, 3) (5, 3) (5, 3) (5, 3) (5, 0) (5, 3)
(5, 3) (5, 3) (5, 3) (5, 3) (5, 3) (5, 3) (5, 3) (5, 3) (0, 3) (5, 3) (5, 0) (5, 0) (0, 0) (0, 0) (0, 0) (5, 0) (5, 3) (5, 3) (0, 3) (0, 3) (0, 3) (5, 3) (5, 3) (5, 3)
(5, 3) (5, 0) (0, 0) (0, 3) (3, 0) (3, 3) (3, 3) (3, 3) (3, 3) (5, 3) (5, 3) (5, 3) (0, 3) (0, 3) (5, 3) (5, 3) (5, 3) (5, 3) (5, 3) (5, 3) (5, 3) (5, 3) (5, 3) (5, 3)
(5, 3) (5, 0) (2, 3) (2, 0) (2, 0) (0, 2) (5, 2) (5, 2) (4, 3)
Result: Success (limit=2000, count=105, history=54)
```



```python
if __name__ == "__main__":
    ### 1 Basic testing of methods and operations
    if True:
        JUG_CFG = [5,3]  # (Die Hard movie version)
        s = setup_jugs()
        print(s)
        # test fillling
        s = fill(s, 0)
        print(s)
        assert s == (5, 0)
        s = pour(s, 0, 1)
        print(s)
        assert s == (2, 3)
        print(s)
        # test emtpy
        s = empty(s, 1)
        print(s)
        assert s == (2, 0)
        # test pour / leftover actions
        s = pour(s, 0, 1)
        assert s == (0, 2)
        print (s)
        s = fill(s, 0)
        print(s)
        assert s == (5, 2)
        s = pour(s, 0, 1)
        print(s)
        assert s == (4, 3)
        s = empty(s, 1)
        print(s)
        assert s == (4, 0)


    ### 2 Solve using a pre-defined sequence of actions
    if True:
        action_calls = {
            'fill': fill,
            'empty': empty,
            'pour': pour
        }

        print('Doing sequence 1 ...')
        ### 3 Sequence 1 of moves
        actions = [
            # tuples, string of method to call then arguments to call
            ('fill',  (0,)),
            ('pour',  (0, 1)),  # (5,0) poor 1 into 2
            ('empty', (1,)),  # (2,3) empty 2
            ('pour',  (0, 1)),  # (2,0) tranfer from 1 to 2
            ('fill', (0,)),  # (0,2) fill jug 1
            ('pour', (0, 1)),
            ('empty', (1,)),
            # TODO: missing move - see header for sequence.
            # result should be (4,0)
        ]

        # execute the sequence of actions
        JUG_CFG = [5,3]  # (Die Hard movie version)
        s = setup_jugs()
        for fn, args in actions:
            #print('Calling...', fn, 'with', args, 'on', s)
            s = action_calls[fn](s, *args)
            print(s)
        print('Done')

    ### 4 Solve using sequence 2
    if True:
        action_calls = {
            'fill': fill,
            'empty': empty,
            'pour': pour
        }

        print('Doing sequence 2 ...')
        actions = [
            # tuples, string of method to call then arguments to call
            ('fill',  [1]),     # fill jug 2 => (0,3)
            ('pour', [1, 0]),  # transfer 2 to jug 1 => (3,0)
            ('fill', [1]),
            ('pour', [1, 0]),
            ('empty', [0]),
            ('pour', [1, 0]),
            ('fill', [1]),
            ('pour', [1, 0]),
            ###TODO: complete the sequence
            # result should be (4,0)
        ]
        # run sequence of actions
        JUG_CFG = [5,3]  # (Die Hard movie version)
        s = setup_jugs()
        for fn, args in actions:
            #print('Calling...', fn, 'with', args, 'on', s)
            s = action_calls[fn](s, *args)
            print(s)
        print('Done')

    # Random choice from all possible actions for a fixed problem
    if True:
        # There is a set of six unique actions to choose from
        actions = [
```

```python
    # all possible fill's
    (fill, [0]),
    (fill, [1]),
    # all possible pour's
    (pour, [0, 1]),
    (pour, [1, 0]),
    # all possble empty's
    (empty, [0]),
    (empty, [1]),
]
# Notes:
# - We exclude pour 0->0 and 1->1 as they pointless
# - Some actions might have no effect (empty if already empty)
#   but we are not making conditional actions (only naive ones)

from random import choice, seed
#seed(1234)

# For the Die Hard 3 movie two-jug problem ...
JUG_CFG = [5, 3]
s = setup_jugs()
s_end = (4, 0)

###TODO: use a list of valid end_states, not just one
end_states = [(4,0), (4,1), (4,2), (4,3)]

status = 'searching'
count = 0
limit =4000
history = []  # history of moves taken

# Search loop
print('Trying a random action search:')
while status == 'searching':

    # select a random action to try
    fn, args = choice(actions)
    new_s = fn(s, *args)

    # print(str(fn.__name__), args, 'on', s, '=>', new_s)  # details
    # print('.', end='')  # progress dots ...
    print(new_s, end=' ')  # verbose

    # if move outcome state is valid (not None) keep it
    if new_s and new_s != s:
        s=new_s
        history.append((fn, args))
        if new_s in end_states:
            status = 'Success'

    # count and stop test
    count += 1
    if count >= limit:
        status = 'Hit limit'

print()
print('Result: %s (limit=%d, count=%d, history=%d)' % (status, limit, count, len(history)))
```