Task 6: Graphs, Search and Rules

Towers of Hanoi: Random Search Results

Running the Towers of Hanoi random search code 10 times for Step 3 of the Tutorial, the program found successful patterns on:

- The 6th attempt: 32 attempted moves, 8 valid moves: [(0, 1), (0, 2), (2, 0), (1, 2), (2, 0), (0, 2), (0, 1), (2, 1), (0, 2), (1, 2), (2, 1), (2, 0), (0, 2), (1, 2), (1, 0), (2, 1), (0, 2), (1, 2)]
- The 8th attempt: 33 attempted moves, 12 valid moves: [(0, 1), (1, 2), (2, 0), (0, 2), (0, 1), (2, 1), (0, 2), (1, 2), (1, 0), (2, 1), (0, 2), (1, 2)]
- The 10th attempt: 55 attempted moves, 22 valid moves: [(0, 1), (0, 2), (2, 0), (1, 2), (2, 0), (0, 1), (1, 2), (2, 0), (0, 1), (1, 0), (0, 2), (0, 1), (2, 0), (0, 1), (1, 0), (0, 1), (1, 2), (2, 1), (0, 2), (1, 0), (1, 2), (0, 2)]

Doing a second run, the program found successful patterns on:

- The 3rd attempt: 55 attempted moves, 30 valid moves: [(0, 2), (2, 0), (0, 2), (2, 1), (1, 2), (2, 0), (0, 2), (0, 1), (2, 0), (0, 2), (2, 1), (0, 2), (1, 0), (0, 2), (2, 0), (0, 1), (2, 0), (1, 0), (0, 2), (2, 0), (0, 1), (0, 2), (1, 0), (0, 2), (2, 0), (1, 2), (2, 1), (1, 2), (0, 1), (1, 2)]
- The 7th attempt: 77 attempted moves, 38 valid moves: [(0, 1), (0, 2), (1, 2), (2, 0), (0, 2), (0, 1), (2, 1), (1, 2), (1, 0), (2, 1), (2, 0), (1, 2), (2, 1), (0, 2), (1, 0), (0, 2), (2, 0), (0, 1), (1, 0), (2, 1), (1, 2), (0, 2), (0, 1), (1, 0), (2, 1), (1, 2), (0, 1), (2, 0), (0, 2), (1, 0), (2, 0), (2, 1), (0, 1), (0, 2), (1, 0), (1, 2), (0, 1), (1, 2)]
- The 9th attempt: 38 attempted moves, 19 valid moves: [(0, 1), (0, 2), (2, 0), (1, 0), (0, 2), (0, 1), (2, 1), (0, 2), (1, 0), (0, 1), (1, 2), (1, 0), (2, 0), (0, 2), (0, 1), (2, 0), (1, 2), (0, 1), (1, 2)]

Towers of Hanoi: Better Random Search

Running the Towers of Hanoi random search code that avoids state A → state B → state A moves for Step 4 of the Tutorial, the program found successful patterns on:

- The 1st attempt: 34 attempted moves, 15 valid moves: [(0, 1), (1, 2), (2, 0), (0, 1), (1, 2), (0, 1), (2, 0), (0, 1), (0, 2), (1, 2), (1, 0), (2, 1), (0, 2), (1, 0), (0, 2)]
- The 2nd attempt: 27 attempted moves, 12 valid moves: [(0, 1), (0, 2), (1, 0), (2, 1), (0, 1), (0, 2), (1, 2), (1, 0), (2, 1), (0, 2), (1, 0), (0, 2)]
- The 7th attempt: 30 attempted moves, 11 valid moves: [(0, 1), (1, 2), (0, 1), (2, 0), (0, 1), (0, 2), (1, 2), (1, 0), (2, 1), (0, 2), (1, 2)]
- The 10th attempt: 86 attempted moves, 39 valid moves: [(0, 1), (1, 2), (0, 1), (2, 0), (1, 2), (0, 2), (2, 1), (2, 0), (1, 2), (0, 1), (2, 0), (0, 1), (1, 2), (1, 0), (2, 1), (0, 2), (1, 2), (0, 1), (2, 1), (1, 0), (0, 2), (1, 0), (2, 0), (2, 1), (0, 2), (1, 0), (2, 1), (0, 2), (1, 2), (2, 0), (2, 1), (0, 1), (0, 2), (1, 2), (1, 0), (2, 1), (0, 2), (1, 0), (0, 2)]

On a second run, the program found successful patterns on:

- The 6th attempt: 34 attempted moves, 17 valid moves: [(0, 2), (2, 1), (1, 0), (0, 2), (0, 1), (2, 0), (0, 1), (0, 2), (1, 0), (0, 2), (2, 1), (1, 0), (0, 2), (2, 1), (1, 0), (1, 2), (0, 2)]

- The 7th attempt: 44 attempted moves, 21 valid moves: [(0, 1), (0, 2), (1, 2), (0, 1), (2, 0), (2, 1), (0, 2), (1, 0), (2, 0), (0, 1), (1, 2), (0, 1), (2, 0), (0, 1), (1, 2), (1, 0), (2, 0), (1, 2), (0, 1), (0, 2), (1, 2)]
- The 8th attempt: 51 attempted moves, 21 valid moves: [(0, 1), (0, 2), (1, 2), (2, 0), (0, 1), (2, 0), (1, 0), (0, 2), (2, 1), (0, 2), (1, 0), (2, 1), (0, 2), (2, 1), (0, 2), (1, 0), (0, 2), (2, 1), (1, 0), (1, 2), (0, 2)]
- The 9th attempt: 22 attempted moves, 7 valid moves (the optimal solution): [(0, 2), (0, 1), (2, 1), (0, 2), (1, 0), (1, 2), (0, 2)]
- The 10th attempt: 54 attempted moves, 22 valid moves: [(0, 1), (0, 2), (1, 2), (2, 0), (0, 1), (1, 2), (0, 1), (2, 0), (0, 1), (1, 2), (2, 0), (2, 1), (0, 2), (1, 0), (2, 0), (1, 2), (0, 1), (1, 2), (2, 0), (0, 1), (0, 2), (1, 2)]

Table 1: Comparison of the results of the pure random and non-reversing random searches

	Random	Random and Non-Reversing
Success Rate	6/20	9/20
Average attempted moves	42.4	48.3
Average valid moves	21.5	22

Compared to the pure random search, the non-reversing random search had a higher success rate for finding valid patterns, a marginally better average number of attempted moves, and almost the same average number of valid moves taken, making it overall the better algorithm for completing the Towers of Hanoi puzzle of the two.

Towers of Hanoi Extension: Random Search with History

I added the code to get the random search to check the proposed state against a cache of previous states it has already been in. The program found successful patterns on:

- The 1st attempt: 16 attempted moves, 8 valid moves: [(0, 2), (0, 1), (2, 1), (0, 2), (1, 2), (2, 0), (1, 2), (0, 2)]
- The 6th attempt: 39 attempted moves, 12 valid moves: [(0, 1), (1, 0), (0, 2), (0, 1), (2, 0), (0, 1), (0, 2), (1, 2), (2, 0), (1, 2), (0, 1), (1, 2)]
- The 8th attempt: 72 attempted moves, 21 valid moves: [(0, 1), (1, 0), (0, 2), (0, 1), (2, 1), (1, 0), (1, 2), (0, 1), (1, 2), (0, 1), (2, 1), (1, 0), (2, 1), (0, 1), (1, 2), (1, 0), (2, 0), (1, 2), (0, 1), (0, 2), (1, 2)]

On a second run, the program found successful results on:

- The 2nd attempt: 91 attempted moves, 14 valid moves: [(0, 2), (2, 0), (0, 1), (0, 2), (1, 0), (2, 1), (0, 2), (2, 1), (0, 2), (1, 2), (2, 0), (1, 2), (0, 1), (1, 2)]
- The 3rd attempt: 19 attempted moves, 9 valid moves: [(0, 1), (0, 2), (1, 0), (2, 1), (0, 1), (0, 2), (1, 0), (1, 2), (0, 2)]
- The 5th attempt: 59 attempted moves, 21 valid moves: [(0, 1), (1, 2), (0, 1), (2, 0), (1, 2), (0, 1), (1, 2), (0, 1), (2, 0), (2, 1), (0, 1), (1, 2), (1, 0), (2, 1), (1, 0), (1, 2), (0, 2), (0, 1), (2, 0), (1, 2), (0, 2)]
- The 7th attempt: 66 attempted moves, 17 valid moves: [(0, 1), (1, 2), (0, 1), (2, 1), (1, 0), (1, 2), (0, 1), (1, 2), (0, 1), (2, 1), (2, 0), (1, 2), (2, 0), (1, 2), (0, 1), (0, 2), (1, 2)]

- The 10th attempt: 26 attempted moves, 11 valid moves: [(0, 1), (0, 2), (1, 0), (2, 1), (0, 1), (0, 2), (1, 2), (2, 0), (1, 2), (0, 1), (1, 2)]

	Random	Random, Non-Reversing	Random, Non-Repeating
Success Rate	6/20	9/20	8/20
Average attempted moves	42.4	48.3	48.5
Average valid moves	21.5	22	14.1

Introducing the check against past states had minimal effect on its success rate or the average number of attempted moves when it did find a successful pattern, but substantially improved the average number of average valid moves required to complete the puzzle, cutting it by a third.

Towers of Hanoi: Recursion

I completed the Towers of Hanoi extension using the code provided in the instructions and tested it against the default n = 3 scenario and an n =5 scenario.

```
C:\Users\Sam Huffer\Coding\COS30002 - 101633177\06 - Lab - Graphs, Search and Rules
>python towers_of_hanoi.py
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
```

Figure 1: Towers of Hanoi recursion with n = 3

```
C:\Users\Sam Huffer\Coding\COS30002 - 101633177\06 - Lab - Graphs, Search and Rules
>python towers_of_hanoi.py
Generating set of moves for 5 (2^n - 1 = 31) disks using recusion ...
- moves (31): [(0, 2), (0, 1), (2, 1), (0, 2), (1, 0), (1, 2), (0, 2), (0, 1), (2,
1), (2, 0), (1, 0), (2, 1), (0, 2), (0, 1), (2, 1), (0, 2), (1, 0), (1, 2), (0, 2),
 (1, 0), (2, 1), (2, 0), (1, 0), (1, 2), (0, 2), (0, 1), (2, 1), (0, 2), (1, 0), (1
, 2), (0, 2)]
    *|*          |           |
   **|**         |           |
  ***|***        |           |
 ****|****       |           |
*****|*****      |           |
> Moving from 0 to 2
```

Figure 2: Towers of Hanoi recursion with n = 5

Water Jug Problem: Asserts

I fixed the assert values such that they would be appropriate for the changes made to the contents of the jugs:

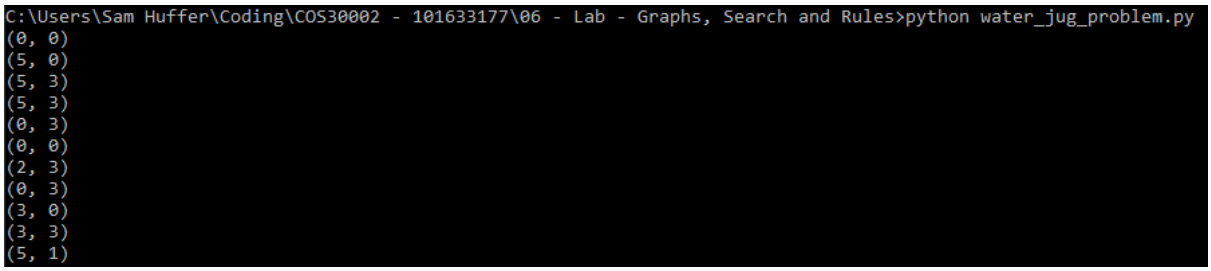

Figure 3: The results of the fixed asserts of the water jug problem.

Water Jug Problem: Solving with a Sequence

I added the last pour statement to sequence 1 such that the sequence would leave the 5-litre jug filled with 4 litres:

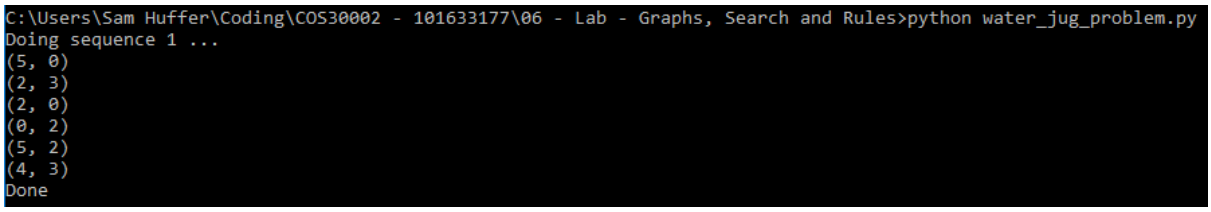

Figure 4: The results of sequence 1 being completed.

Furthermore, I added the required statements to sequence 2 to achieve the same result:

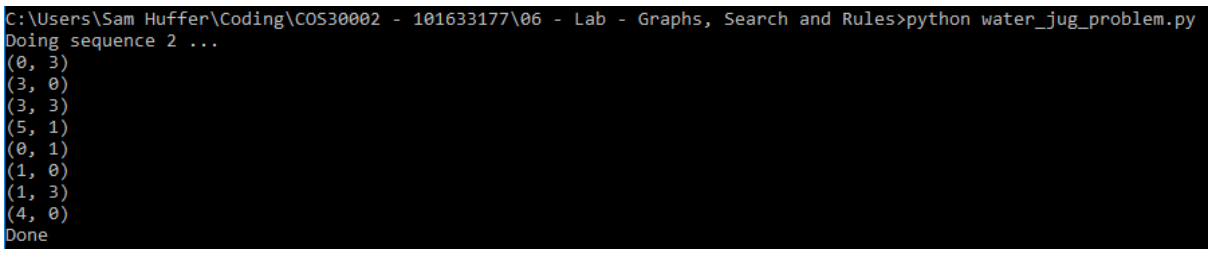

Figure 5: The results of sequence 2 being completed.

Water Jug Problem: Random Search

Running the code initially, it kept hitting the search limit, so I doubled the search limit to see how that would affect the results. It kept hitting the limit without a successful result, and I must have tried it 20 or 30 times with no successes. It kept making moves that reverted it back to a (0, 0) state:

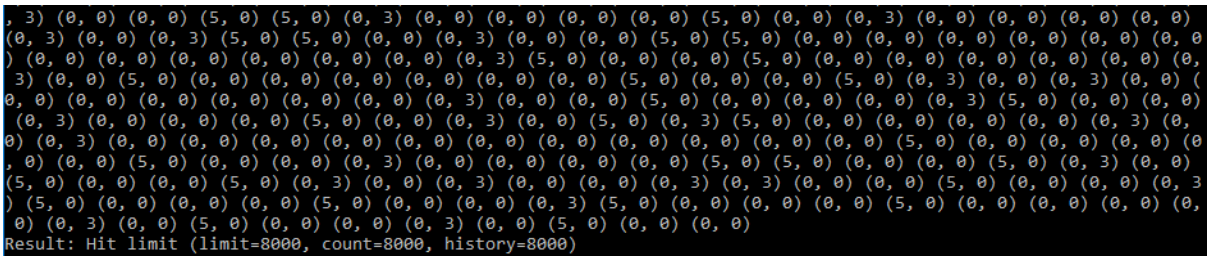

Figure 6: An unsuccessful random search with limit 8000. Note the prevalence of (0, 0) states.

Water Jug Problem: Better Random Search

I added the code to have the program check for the full set of win states, but it had no more success than the random search looking only for a (4, 0) state. Adding the check against doing moves that do not change the game state finally resulted in successes:

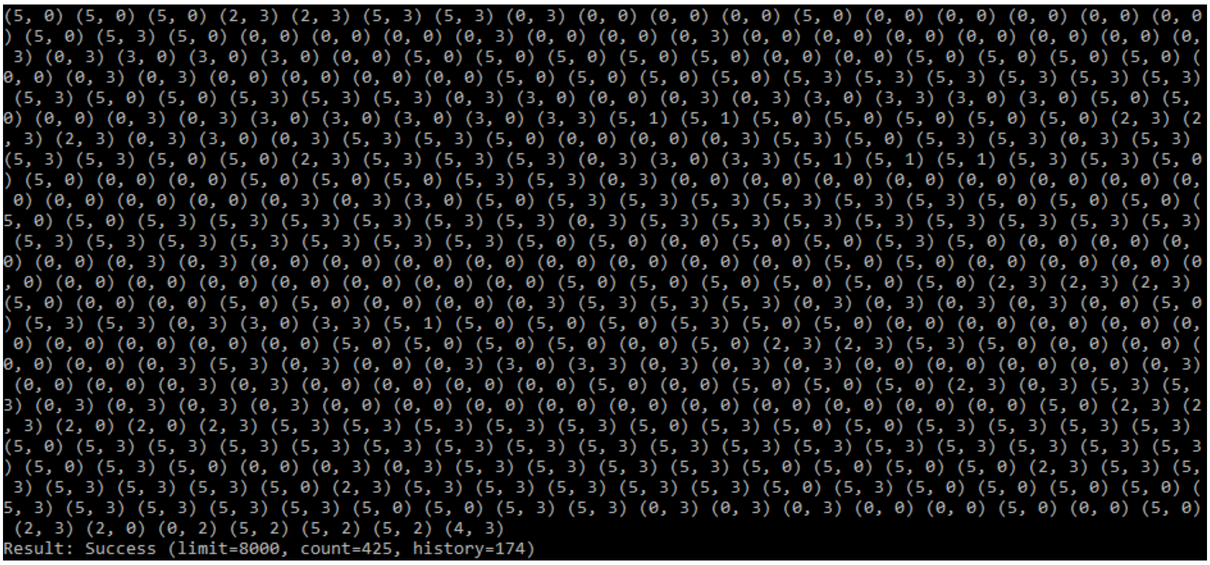

Figure 7: A successful non-repeating random search.

Running the program 20 times yielded the following data:

Table 2: The results of 20 non-repeating random searches.

Success Rate	Average Count on Success	Average History on Success	Average proportion of invalid moves
16/20	2544.4	1126.4	55.7 %

Adding a cache of all past moves and preventing it from repeating a previous move yielded the following data when run 20 times:

Table 3: The results of 20 non-repeating random searches.

Success Rate	Average Count on Success	Average History on Success	Average proportion of invalid moves
11/20	52.7	9.7	81.6%

While the success rate was cut by a third, the average number of attempted moves on a successful result came down dramatically, as did the average number of valid moves on a successful result, vastly improving the quality of results that were successful. Matching the latter, the average number of invalid moves (which now included already visited states) went up by more than half, which isn't a big concern in this instance as we don't want to be going around in circles.