Assignment 3: Blocks World

Group Members & Work Done:

Ryan Sippy, Nathan Grilliot, Derek Corniello

All team members contributed an equal measure

Honor Statement:

In completing this assignment, all team members have followed the honor pledge specified by the instructor for this course.

Bibliography:

None

Runs:

Run 1:

Start: [on, a, b], [on, b, "table"], [on, c, d], [clear, c], [clear, a], [on, d, "table"]

Goal: [on, d, a], [on, a, c], [on, c, b], [on, b, "table"], [clear, d]

Final Path:

```
Path =
[[[on, a, b], [on, b, "table"], [on, c, d], [clear, c], [clear, a], [on, d, "table"]],
[[on, c, "table"], [clear, d], [on, a, b], [on, b, "table"], [clear, c], [clear, a], [on, d, "table"]],
[[on, a, "table"], [clear, b], [on, c, "table"], [clear, d], [on, b, "table"], [clear, c], [clear, a], [on, d, "table"]],
[[on, b, c], [on, a, "table"], [clear, b], [on, c, "table"], [clear, d], [clear, a], [on, d, "table"]],
[[on, d, a], [on, b, c], [on, a, "table"], [clear, b], [on, c, "table"], [clear, d]],
[[on, b, "table"], [clear, c], [on, d, a], [on, a, "table"], [clear, b], [on, c, "table"], [clear, d]],
[[on, c, b], [on, b, "table"], [clear, c], [on, d, a], [on, a, "table"], [clear, d]],
[[on, d, "table"], [clear, a], [on, c, b], [on, b, "table"], [clear, c], [on, a, "table"], [clear, d]],
[[on, a, c], [on, d, "table"], [clear, a], [on, c, b], [on, b, "table"], [clear, d]],
[[on, d, a], [on, a, c], [on, c, b], [on, b, "table"], [clear, d]]]
```

Discussion:

In our first run, the BlocksWorld code was able to successfully move the blocks to their goal state in a reasonable number of moves. From closer examination, the moves taken seem to be logical, with all moves bringing the state of the blocks closer to the goal state. The program uses its table locations by moving clear blocks to the table or to another clear block. This reorders the blocks into one stack successfully, reaching the final goal.

Run 2:

Start: [on, a, b], [on, b, c], [on, c, d], [clear, a], [on, d, "table"]

Goal: [on, d, a], [on, a, c], [on, c, b], [on, b, "table"], [clear, d]

Final Path:

```
Path =
[[[on, a, b], [on, b, c], [on, c, d], [clear, a], [on, d, "table"]],
[[on, a, "table"], [clear, b], [on, b, c], [on, c, d], [clear, a], [on, d, "table"]],
[[on, b, "table"], [clear, c], [on, a, "table"], [clear, b], [on, c, d], [clear, a], [on, d, "table"]],
[[on, c, "table"], [clear, d], [on, b, "table"], [clear, c], [on, a, "table"], [clear, b], [clear, a], [on, d, "table"]],
[[on, b, c], [on, c, "table"], [clear, d], [on, a, "table"], [clear, b], [clear, a], [on, d, "table"]],
[[on, d, a], [on, b, c], [on, c, "table"], [clear, d], [on, a, "table"], [clear, b]],
[[on, b, "table"], [clear, c], [on, d, a], [on, c, "table"], [clear, d], [on, a, "table"], [clear, b]],
[[on, c, b], [on, b, "table"], [clear, c], [on, d, a], [clear, d], [on, a, "table"]],
[[on, d, "table"], [clear, a], [on, c, b], [on, b, "table"], [clear, c], [clear, d], [on, a, "table"]],
[[on, a, c], [on, d, "table"], [clear, a], [on, c, b], [on, b, "table"], [clear, d]],
[[on, d, a], [on, a, c], [on, c, b], [on, b, "table"], [clear, d]]]
```

Discussion:

In our next run, our BlocksWorld code correctly moved the blocks to the goal state in a decent number of moves. Upon reviewing the moves, it shows that our algorithm is not necessarily optimal, which can be attributed to the depth limitation we gave the code. However, the program created still arrives at the final goal state, so we know that while it may not be optimal, it will always be correct. The logic of how it finds the next state is, like above, using the table locations and clearing blocks to make another state closer to the final goal state.

Run 3:

Start: [on, d, a], [on, a, "table"], [on, b, c], [clear, b], [clear, d], [on, c, "table"]

Goal: [on, d, a], [on, a, c], [on, c, b], [on, b, "table"], [clear, d]

Final Path:

```
Path =
[[[on, d, a], [on, a, "table"], [on, b, c], [clear, b], [clear, d], [on, c, "table"]],
[[on, b, "table"], [clear, c], [on, d, a], [on, a, "table"], [clear, b], [clear, d], [on, c, "table"]],
[[on, d, "table"], [clear, a], [on, b, "table"], [clear, c], [on, a, "table"], [clear, b], [clear, d], [on, c, "table"]],
[[on, c, b], [on, d, "table"], [clear, a], [on, b, "table"], [clear, c], [on, a, "table"], [clear, d]],
[[on, a, c], [on, c, b], [on, d, "table"], [clear, a], [on, b, "table"], [clear, d]],
[[on, d, a], [on, a, c], [on, c, b], [on, b, "table"], [clear, d]]]
```

Discussion:

In our final test run, our BlocksWorld code was successful in finding a path from the start state to the goal state. When looking at the final path, the path seems logical in how it moves blocks to reach the goal state. It searches through various states to find the goal state by moving clear blocks to the table or moving blocks onto each other in its search for the goal state.

Final Discussion:

Overall, our BlocksWorld code, though not optimal, is successful in finding a path from a start state to a goal state. Our implementation uses a limited depth, Depth-First Search to search through various possible states to find a path to the goal state. It also uses permutation checking to make sure that it is not evaluating the same state multiple times. This was added to make sure that the implementation does not get stuck in an infinite loop of repeatedly checking the same state. The speed of our implementation is tied to how many states that it must be searched upon to find the goal state. This makes sense due to our implementation making use of DFS which has a time complexity of O(b^m) which is related to the depth of the solution.