The second part of the Rubik's cube implementation builds on top of the code from part 1. Some updates needed to be made in order to accommodate the new requirements.
To begin with, the print_seq class method enables the output of multiple cube representations with three per line, as desired. This has been accomplished by creating a new matrix which holds the information of up to three normalized states using the append_matrix helper function. Next is the update class method which updates the initial state string and normalized state inside the ids method and the dls command logic. The expand method is also used to return all the successors of a given node, which is obtained by applying every available move to the state and returning a list of all its successors. Finally, two helper functions, heuristic and heuristic_mapping, were included to aid in the implementation of the A* algorithm which will be discussed later.
As for the required search functions, we begin with the breadth-first search. The history of visited states is recorded in the set closed, while the fringe is composed of all the states we could visit next, along with the paths needed to reach them represented as the required move sequence. Given the fringe architecture, a separate set named open was added in order to more easily check if a state is already present in the fringe. A successor is added into the fringe only if it has not been visited before and if it is not already present in the fringe. The expanded node from the fringe is always the first element, like a FIFO queue. The function returns the sequence of moves required to reach a goal state along with the total number of visited nodes if a solution exists, and None if it does not.
In order to be usable as part of the ids method as well, the initial state is updated inside of the command logic for depth-limited search rather than inside the method. The algorithm for dls is very similar to the one for bfs. This time around, the fringe also includes the depth of a given node, along with the state and sequence path. As for the next expanded node, this time we are taking the last state added in the fringe, like a LIFO queue. What's more, current nodes are added into the closed set and successor nodes into the fringe only if the current depth is below that of the limit. Upon testing with different depth limits, I realized that keeping the closed set implementation identical to that of the other search algorithms, so only keeping track of the visited states, resulted in the algorithm not returning a solution unless one was found exactly at the depth limit. That was because once a state was visited at a certain depth, the algorithm would not revisit the same configuration at lower depths, since it was already in the closed set. To correct this, the closed set is now comprised of both the current node and its respective depth. As such, in order for a successor to be added to the fringe, other than the successor not being present in the open set, the tuple of the successor and its respective depth must not be present in the closed set either. This ensures against a given configuration at a lower depth being skipped over.
The iterative deepening search is simply iteratively calling the dls method with one depth limit at a time, up until the overall depth limit inputted in the command call. As opposed to the dls command, the initial state is now imputed inside the ids methods. The range of the depth limit iterations goes to dl+1 in order to accommodate the mechanism of the dls.
Moving on to the A* informed search, we must first define the heuristic used to estimate the distance to the goal. To help with that, the heuristic_mapping function takes a state and maps each cubie to its corresponding corner in 3 dimensions. The range of the corner coordinates is between 1 and 0. The heuristic function creates two of these dictionaries, one for the default

state, and one for the current state. The Manhattan distance is then computed and summed for all 8 corners, and then divided by 4 in order to be admissible.
The astar method's fringe now contains the sum of the number of moves from the initial state to the current state and Manhattan distance to the goal, the number of moves from the initial state to the current state by itself, the current state, and the sequence of moves. The fringe is sorted at each iteration, and the node with the lowest f is expanded. The rest of the functionality is similar to that of the previous search algorithms.
Moving on to time and space complexity analysis, the breadth-first search processes all of the nodes above the shallowest solution of depth s, leading to a search time of O(b^s). The bfs also keeps in memory all of the nodes until roughly the last tier, resulting in a space complexity of O(b^s). One of the main advantages of bfs is that if all costs are 1, which is the case in our exercise, the solution will always be optimal and complete. Conversely, the depth-limited search expands some left prefix on the tree, meaning that it could process it in its entirety and leading to a time complexity of O(b^l), where l is the depth limit. What's more, dls will find the leftmost solution regardless of depth or cost, making it not optimal. As long as cycles are prevented, which is the case in this implementation, the search satisfies completeness. The main advantage of dls is its space complexity, which only contains the sibling on the path to the root and has the complexity of O(b*l). The iterative deepening method combines bfs's time and optimality advantages with dls's space advantage. On the other hand, the time and space complexities of the A* algorithm are harder to analyze. In a worst-case scenario, the algorithm has the same time and space complexities as the bfs. However, the choice of heuristic can drastically improve A*'s results. In our case, the heuristic chosen is admissible and consistent, meaning that it will always finish faster than bfs.
Given our results, the search implementations perform largely as expected, especially when it comes to space complexity. As for the time complexities, we can spot the exponential behavior when the optimal solution is found at increasing depths. Throughout all 3 different initial state configurations, the A* algorithm has dramatically outperformed the other 3. We can see that in all 3 examples, the dls algorithm returns a non-optimal solution, with all others computing the optimal one. In order to verify the aforementioned issue of the dls algorithm not returning solutions at lower depths, another dls call with a depth limit for which a solution is not available is included, and works as expected. In 2 out of the 3 examples, the bfs algorithm outperforms the dls significantly. For the one that is not the case, it can be attributed to the fact that the solution happened to be "leftmost" when compared to the optimal one. It is also noteworthy to mention how the ids is much more efficient than both bfs and dls throughout all the examples, and is even comparable to A*, which illustrates the advantage of combining the benefits of both bfs and dls.

Results:

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh bfs "L D' R' F R D'"
U F' R' U R U'
   BW              GB              GB
   GW              WW              OO
OY RR BB OY     RR BB OY OY     RW BY GY OY
WR BY OO WG     WR BY OO WG     WW BB YO WG
   YG              YG              RR
   RG              RG              RG


   GW              OG              OO
   OO              OW              OO
RW BB YO GY     BB YO GY RW     BB YY GG WW
WW BO GY RG     WW BO GY RG     WW BB YY GG
   RY              RY              RR
   RB              RB              RR


   OO
   OO
WW BB YY GG
WW BB YY GG
   RR
   RR


68516
7.88
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh dls "L D' R' F R D'" 8
B' F L D' B' L D B'
   BW              WO              WO
   GW              GW              RY
OY RR BB OY     RY RR BB YG     RY BR GB YG
WR BY OO WG     GR BY OW OW     GG YR WW OW
   YG              YG              OB
   RG              OB              OB


   WO              WO              RG
   GY              GY              GY
GR WR GB YO     GR WR GB YO     BR WR GW OY
GY RR WW OO     RR WW OO GY     YR WW OO YG
   BB              BB              BB
   YB              BY              OB


   GG              GG              YY
   YY              YY              YY
YB RR GW OO     YB RR GW OO     BB RR GG OO
RR GW OO YB     YB RR GW OO     BB RR GG OO
   WB              WW              WW
   WB              BB              WW


2085844
14.2
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh dls "L D' R' F R D'" 7
D L' D' B D B'
   BW              BW              RW
   GW              GW              WW
OY RR BB OY     OY RR BB OY     YG RR BB OG
WR BY OO WG     WG WR BY OO     OW GR BY OB
   YG              RY              OY
   RG              GG              YG


   RW              BB              BB
   WW              WW              WW
YG RR BB OG     WG RR BY OO     WG RR BY OO
GR BY OB OW     RR BY OO WG     WG RR BY OO
   YG              YG              YY
   OY              YG              GG


   WW
   WW
GG RR BB OO
GG RR BB OO
   YY
   YY


2053702
14.41
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh ids "L D' R' F R D'" 20
Depth: 0 d: 1
Depth: 1 d: 13
Depth: 2 d: 157
Depth: 3 d: 1479
Depth: 4 d: 12073
Depth: 5 d: 86414
Depth: 6 d: 359500
IDS found a solution at depth 6
D L' D' B D B'
   BW              BW              RW
   GW              GW              WW
OY RR BB OY     OY RR BB OY     YG RR BB OG
WR BY OO WG     WG WR BY OO     OW GR BY OB
   YG              RY              OY
   RG              GG              YG


   RW              BB              BB
   WW              WW              WW
YG RR BB OG     WG RR BY OO     WG RR BY OO
GR BY OB OW     RR BY OO WG     WG RR BY OO
   YG              YG              YY
   OY              YG              GG


   WW
   WW
GG RR BB OO
GG RR BB OO
   YY
   YY


459637
2.96
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh astar "L D' R' F R D'"
D L' U' L D L'
   BW              BW              RW
   GW              GW              WW
OY RR BB OY     OY RR BB OY     YG RR BB OG
WR BY OO WG     WG WR BY OO     OW GR BY OB
   YG              RY              OY
   RG              GG              YG


   WW              BW              BW
   RW              BW              BW
OG YG RR BB     OO WG RR BY     OO WG RR BY
OW GR BY OB     WG RR BY OO     OO WG RR BY
   OY              YY              GY
   YG              GG              GY


   WW
   WW
OO GG RR BB
OO GG RR BB
   YY
   YY


1462
1.15
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh bfs "L' B' U' D L' F B"
F F U U F
   GG             GG             GG
   BB             BW             YY
WW RR YY RR    WY OR BY RR    WG OO BY RR
BB OO GG OO    BY OR BG OO    BY RR WG OO
   YY             GY             BB
   WW             WW             WW


   YG             YY             YY
   YG             GG             YY
OO BY RR WG    BY RR WG OO    BB RR GG OO
BY RR WG OO    BY RR WG OO    BB RR GG OO
   BB             BB             WW
   WW             WW             WW


30410
2.69
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh dls "L' B' U' D L' F B" 8
B' B' L' R B' B' D
   GG             BW             WW
   BB             BB             BB
WW RR YY RR    WW RR YG RO    GW RR YB OO
BB OO GG OO    WB OO GG RO    YB OO GW RR
   YY             YY             YY
   WW             GY             GG


   RW             RR             GW
   OB             OO             OO
WB YR YB OB    WB YY GY BB    OB YY GR BW
GY GO GW RW    GY GG WB WW    OY GG WR BW
   RY             RR             RR
   OG             OO             BY


   OO             OO
   OO             OO
BB YY GG WW    BB YY GG WW
YY GG WW BB    BB YY GG WW
   RR             RR
   RR             RR


63655
0.42
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh ids "L' B' U' D L' F B" 20
Depth: 0 d: 1
Depth: 1 d: 13
Depth: 2 d: 157
Depth: 3 d: 1479
Depth: 4 d: 12073
Depth: 5 d: 332
IDS found a solution at depth 5
B' B' D' B
   GG             BW             WW
   BB             BB             BB
WW RR YY RR    WW RR YG RO    GW RR YB OO
BB OO GG OO    WB OO GG RO    YB OO GW RR
   YY             YY             YY
   WW             GY             GG


   WW             WW             BB
   BB             BB             BB
GW RR YB OO    GW RR YB OO    WW RR YY OO
OO GW RR YB    GW RR YB OO    WW RR YY OO
   YG             GG             GG
   YG             YY             GG


14055
0.09
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh astar "L' B' U' D L' F B"
B' B' U' D' R
   GG             BW             WW
   BB             BB             BB
WW RR YY RR    WW RR YG RO    GW RR YB OO
BB OO GG OO    WB OO GG RO    YB OO GW RR
   YY             YY             YY
   WW             GY             GG


   WB             WB             WW
   WB             WB             WW
OO GW RR YB    OO GW RR YB    OO GG RR BB
YB OO GW RR    OO GW RR YB    OO GG RR BB
   YY             YG             YY
   GG             YG             YY


173
0.05
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh shuffle 9
F' R' B F' R' B' B' F R

   GW
   RY
OW GG OR BW
YR YR BW OO
   GY
   BB
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh bfs "F' R' B F' R' B' B' F R"
R' F R F U
   GW             GO             GO
   RY             RB             RW
OW GG OR BW    OW GW RW BW    OG YG RW BW
YR YR BW OO    YR YY OB YO    YG YW BB YO
   GY             GG             OR
   BB             BR             BR


   GG             GG             GG
   RW             GG             GG
OG YR BR WW    OO YY RR WW    YY RR WW OO
YG YR BW OO    YY RR WW OO    YY RR WW OO
   OY             BB             BB
   BB             BB             BB


24933
2.11
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh dls "F' R' B F' R' B' B' F R" 10
B' B' D B' B' L B' D' L'
   GW             YO             BB
   RY             RY             RY
OW GG OR BW    BW GG OG WO    WW GG OY OO
YR YR BW OO    BR YR BW BO    RR YR BO WB
   GY             GY             GY
   BB             WR             WG


   BB             WW             YG
   RY             RY             RY
WW GG OY OO    GW GG OB OO    RW GG OW OB
WB RR YR BO    YB RR YB OB    YB RR YW OO
   WG             WG             WG
   GY             RY             BB


   OG             BY             BY
   BY             BY             BY
YR YG OW OB    RR YG OO BW    RR YG OO BW
BW RR YW OW    BW RR YG OO    RR YG OO BW
   GG             GG             GW
   RB             WW             GW


   YY
   YY
RR GG OO BB
RR GG OO BB
   WW
   WW


6121834
42.52
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh ids "F' R' B F' R' B' B' F R" 10
Depth: 0 d: 1
Depth: 1 d: 13
Depth: 2 d: 157
Depth: 3 d: 1479
Depth: 4 d: 12073
Depth: 5 d: 25707
IDS found a solution at depth 5
L' D B D B
   GW             GW             GW
   RY             YY             YY
OW GG OR BW    WR GG OR BR    WR GG OR BR
YR YR BW OO    OY BR BW OG    OG OY BR BW
   GY             OY             WO
   BB             WB             BY


   RR             RR             YY
   YY             YY             YY
WR GG OY BB    WR GG OY BB    RR GG OO BB
GG OY BB WR    WR GG OY BB    RR GG OO BB
   WO             WW             WW
   WO             OO             WW


39430
0.26
```

```
(base) iustintoader@Iustins-MBP Assignment 3 % sh run.sh astar "F' R' B F' R' B' B' F R"
R' F L D B
   GW             GO             GO
   RY             RB             RW
OW GG OR BW    OW GW RW BW    OG YG RW BW
YR YR BW OO    YR YY OB YO    YG YW BB YO
   GY             GG             OR
   BB             BR             BR


   OO             OO             WW
   WW             WW             WW
YO GG RW BB    YO GG RW BB    OO GG RR BB
GG RW BB YO    YO GG RW BB    OO GG RR BB
   YR             YY             YY
   YR             RR             YY


331
0.12
```