Explain your program

My program builds on top of my submission for assignment 2. A few new commands have been introduced:

- **bfs** – runs bfs search after scrambling the cube using provided string
- **ids**– runs ids search after scrambling the cube using provided string
- **astar**– runs A* search after scrambling the cube using provided string
- **idastar**– runs IDA* search after scrambling the cube using provided string
- **astarMD**– runs A* search after scrambling the cube using provided string but uses Manhattan distance for heuristic
- **createDB –** creates a pattern database for the heuristic function

bfs, ids, and astarMD run exactly as described in the assignment Hand-out.

While researching on how to speed up my algorithm, I came across a few other techniques used by other people. One of them was **IDA***. This algorithm is like Iterative Deepening Search except instead of using limits as a cutoff, we use heuristic of the nodes. The initial value of the heuristic limit is taken as the heuristic of the current state and the I increase it by 0.5 after every iteration. I also learned about **Pattern Databases** that I explain in the heuristics section.

I used knowledge from this paper on various ways to solve a Rubik's cube by Harpreet Kaur to implement my **IDA***: https://www.diva-portal.org/smash/get/diva2:816583/FULLTEXT01.pdf

Note: If the pattern database does not exist in the file system, it takes ~10 minutes to create the database and run the requested command.

Analyzing the time and space complexity of algorithms and comparison

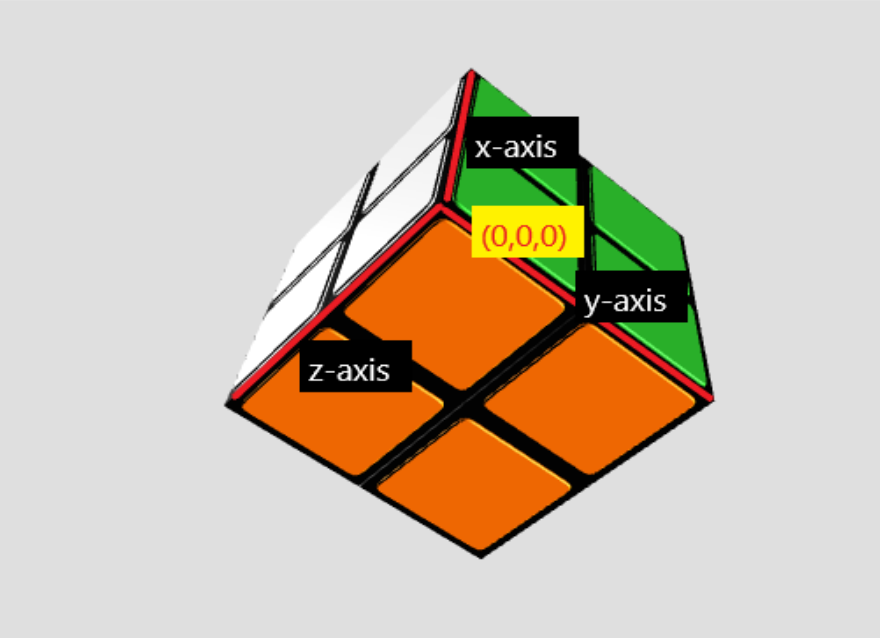
Time complexity of BFS is O(branching factor^{depth}). The branching factor for my algorithm is 12 on the first level and then 10 for subsequent levels as redundant, inverse and complimentary moves are removed. The space complexity will be similar.

Time complexity for my IDS is similar to BFS, O(branching factor^{depth}) but the space complexity will be O(branching factor * depth).

Since astar is best first search, time and space complexity for astar will also be similar to BFS.

Heuristic

My first heuristic function calculates the Manhattan distance using the following coordinate system:-



To find the coordinates, I do some unnecessarily complicated calculations. First, I have declared a few global arrays: corners, cornersIdeal, cornersCoord, cornersIdealSorted. After I find what each corner holds, I create a key by concatenating its 3 colors into a string list and sorting it alphabetically. This way, I'm able to ignore the orientation and yet I can find the corresponding coordinates using indexes. After I find the current coordinates of the corner and where it is supposed to be, I find the Manhattan distance and divide it by 4. I return the sum of the result and the depth of the node passed.

Another thing that I learned during my research was **Pattern Databases** for heuristics. This method performs breadth first search on a solved state and calculates the heuristic for all the permutations of certain pattern. It then indexes these states with their heuristic, which is depth from the solved state. This way, I get the shortest distance to the solution. I stored all this in a python dictionary and then dumped it into a .pkl file using python's pickle library. Not the most space and time efficient method but it works.

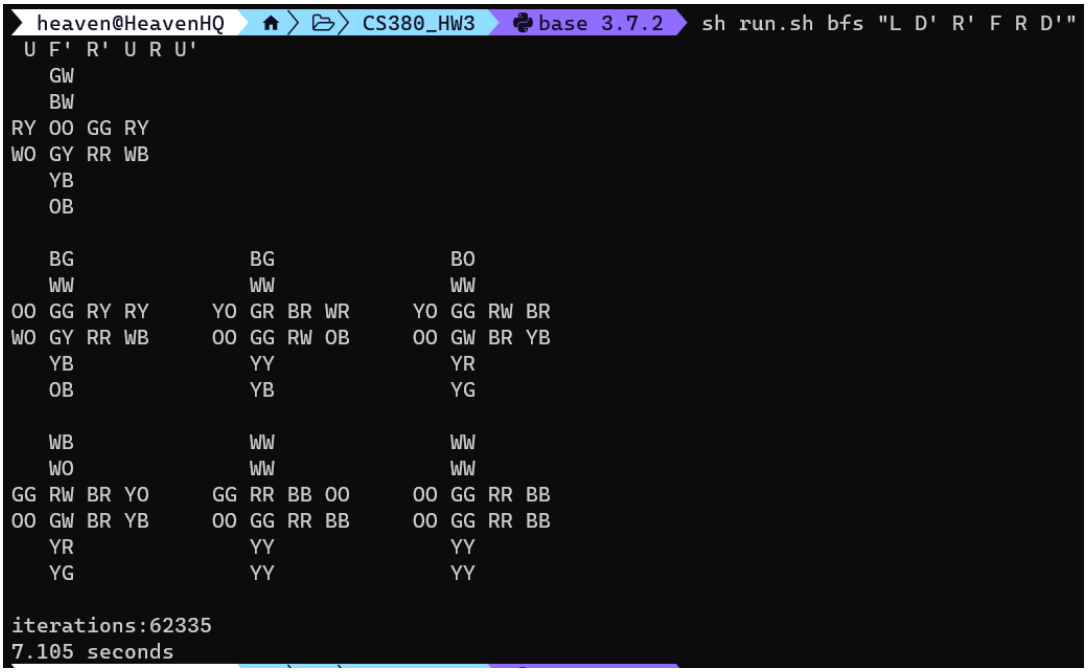
The fixed pattern I used here was adapted from *Richard E. Korf*'s paper "*Finding Optimal Solutions to Rubik's Cube Using Pattern Databases*". Korf calculates every possible legal combination for corners of a 3x3x3 cube and stores the corresponding heuristic to just solve the corners into a database. While this is a sub-pattern in a larger pattern in a 3x3x3, it becomes the main pattern in 2x2x2's case since we don't have any edge cases to worry about. The article can be found here.

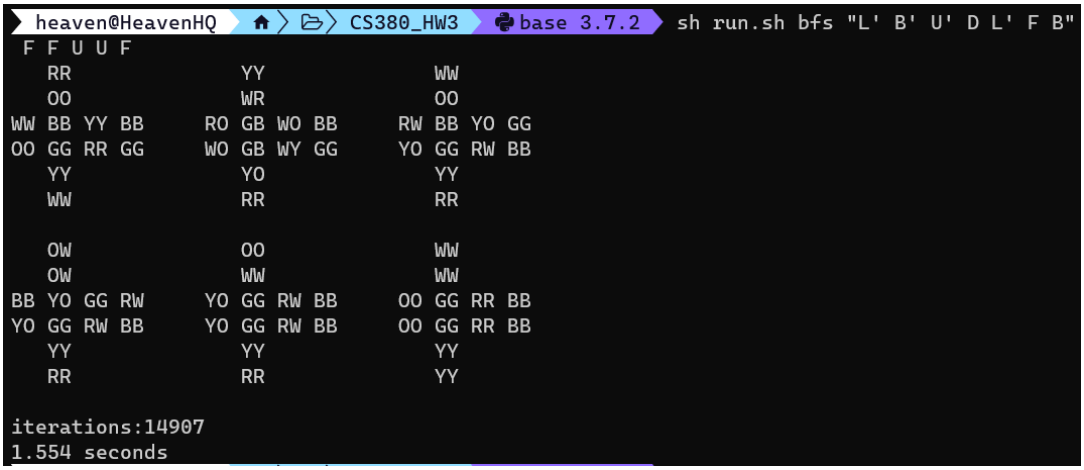
The database is created using create_pattern_database() function. It runs the breadth first search on the solved state as mentioned earlier and checks 3,674,160 nodes, which is the total number of valid states a 2x2x2 can have according to jaapch.net. This is then indexed and stored into a file in the same directory. A database takes ~450 seconds to generate from scratch and ~2 seconds to load into the program. To save space while storing the database, I convert the string representation of the state to an integer representation.

Results showing testing of your routines

For testing, I used the algorithms provided in the hand-in. For my astar, I also used a longer scramble algorithm to test it.

1. BFS





2. IDS

```
heaven@HeavenHQ  CS380_HW3  base 3.7.2  sh run.sh ids "L D' R' F R D'"
Depth: 0 d: 0
Depth: 1 d: 12
Depth: 2 d: 132
Depth: 3 d: 1320
Depth: 4 d: 13200
Depth: 5 d: 131892
Depth: 6 d: 574759
IDS found a solution at depth 6
 D L' D' B D B'
   GW
   BW
RY OO GG RY
WO GY RR WB
   YB
   OB

   RG              WO              OW
   OG              OO              WW
WB YY RR WB     RG WW BB YG     YB OO GG RB
GO GY RB WW     YO GW BR YB     BO GY RG RW
   YB              YR              YB
   OO              RG              RY

   GG              RR              WW
   WW              WW              WW
WB OO GY RR     WO GG RY BB     OO GG RR BB
OO GY RR WB     WO GG RY BB     OO GG RR BB
   YB              YY              YY
   YB              OO              YY

iterations: 721315
9.491 seconds
```

```
heaven@HeavenHQ  CS380_HW3  base 3.7.2  NOTFOUND  sh run.sh ids "L' B' U' D L' F B"
Depth: 0 d: 0
Depth: 1 d: 12
Depth: 2 d: 132
Depth: 3 d: 1320
Depth: 4 d: 13200
Depth: 5 d: 311
IDS found a solution at depth 5
 B' B' D' D' B
   RR              OW              WW
   OO              OO              OO
WW BB YY BB     WW BB YR BG     RW BB YO GG
OO GG RR GG     WO GG RR BG     YO GG RW BB
   YY              YY              YY
   WW              RY              RR

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

iterations: 14975
0.173 seconds
```

3. A* using Pattern Database

```
heaven@HeavenHQ  CS380_HW3  base 3.7.2  sh run.sh astar "L' B' U' D L' F B"
Loading pattern database, please hold....
Database loaded!! Commencing request!
F' F' U U F

   RR              RR              WW
   OO              WR              OO
WW BB YY BB     YO GB WW GG     RW BB YO GG
OO GG RR GG     OO GB WR BB     YO GG RW BB
   YY              YO              YY
   WW              YY              RR

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

iterations:5
0.002 seconds
```

```
heaven@HeavenHQ  CS380_HW3  base 3.7.2  sh run.sh astar "L D' R' F R D'"
Loading pattern database, please hold....
Database loaded!! Commencing request!
U F' R' U R U'

   GW
   BW
RY OO GG RY
WO GY RR WB
   YB
   OB

   BG              BG              BO
   WW              WW              WW
OO GG RY RY     YO GR BR WR     YO GG RW BR
WO GY RR WB     OO GG RW OB     OO GW BR YB
   YB              YY              YR
   OB              YB              YG

   WB              WW              WW
   WO              WW              WW
GG RW BR YO     GG RR BB OO     OO GG RR BB
OO GW BR YB     OO GG RR BB     OO GG RR BB
   YR              YY              YY
   YG              YY              YY

iterations:6
0.002 seconds
```

```
heaven@HeavenHQ  CS380_HW3  base 3.7.2  sh run.sh astar "U R' F' U' F' U' R R F U U"
Loading pattern database, please hold....
Database loaded!! Commencing request!
D U R' B' F' R F U F U R'

   BR              BW              WB
   RW              WO              OW
WY BR BG YO     OR BW BG RY     BW BG RY OR
GO GO YG WW     OO GY GY RG     OO GY GY RG
   YB              YR              YR
   OR              WB              WB

   WR              OB              OG
   OO              OO              WR
BW BB YY BR     WW BB YW RG     YO GY BY RB
OO GW RG RG     YO GW RR BR     WO GB WR GR
   YG              YG              YO
   WY              GY              BW

   OY              WB              WW
   WB              WW              WB
YO GO WB RB     BB OO GR YR     OO GR YR BB
WO GW RY GR     GO GW RB OY     GO GW RB OY
   YG              YG              YG
   BR              RY              RY

   GG              WG              WW
   WW              WG              WW
WB OO GY RR     OO GY RR WB     OO GG RR BB
OO GY RR WB     OO GY RR WB     OO GG RR BB
   YB              YB              YY
   YB              YB              YY

iterations:11
0.003 seconds
```

4. A* only using Manhattan Distance

```
heaven@HeavenHQ  CS380_HW3  base 3.7.2  sh run.sh astarMD "L D' R' F R D'"
U F' R' U R U'

   GW
   BW
RY OO GG RY
WO GY RR WB
   YB
   OB

   BG              BG              BO
   WW              WW              WW
OO GG RY RY     YO GR BR WR     YO GG RW BR
WO GY RR WB     OO GG RW OB     OO GW BR YB
   YB              YY              YR
   OB              YB              YG

   WB              WW              WW
   WO              WW              WW
GG RW BR YO     GG RR BB OO     OO GG RR BB
OO GW BR YB     OO GG RR BB     OO GG RR BB
   YR              YY              YY
   YG              YY              YY

iterations:3168
2.906 seconds
```

```
heaven@HeavenHQ  CS380_HW3  base 3.7.2  sh run.sh astarMD "L' B' U' D L' F B"
F' F' U U F

   RR              RR              WW
   OO              WR              OO
WW BB YY BB     YO GB WW GG     RW BB YO GG
OO GG RR GG     OO GB WR BB     YO GG RW BB
   YY              YO              YY
   WW              YY              RR

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

iterations:295
0.084 seconds
```

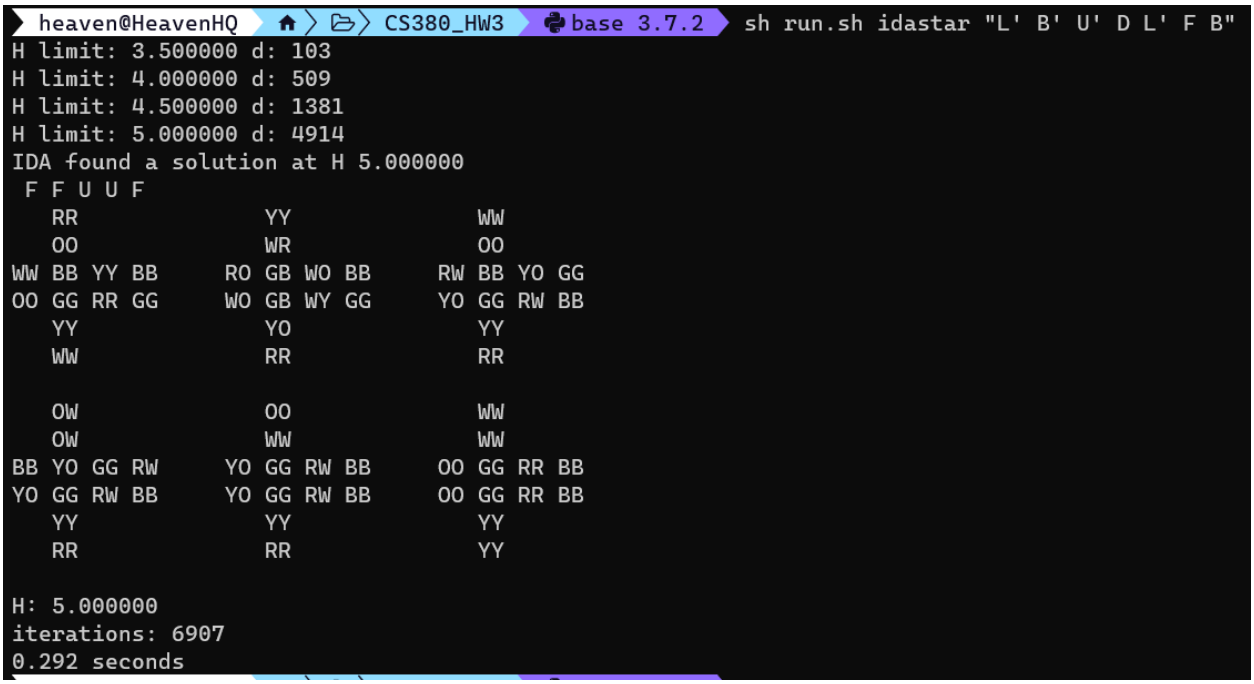
5. IDA*

```
heaven@HeavenHQ  CS380_HW3  base 3.7.2  sh run.sh idastar "L D' R' F R D'"
H limit: 3.500000 d: 201
H limit: 4.000000 d: 757
H limit: 4.500000 d: 2375
H limit: 5.000000 d: 6382
H limit: 5.500000 d: 19293
H limit: 6.000000 d: 50630
IDA found a solution at H 6.000000
 U F' R' U R U'
   GW
   BW
RY OO GG RY
WO GY RR WB
   YB
   OB

   BG              BG              BO
   WW              WW              WW
OO GG RY RY     YO GR BR WR     YO GG RW BR
WO GY RR WB     OO GG RW OB     OO GW BR YB
   YB              YY              YR
   OB              YB              YG

   WB              WW              WW
   WO              WW              WW
GG RW BR YO     GG RR BB OO     OO GG RR BB
OO GW BR YB     OO GG RR BB     OO GG RR BB
   YR              YY              YY
   YG              YY              YY

H: 6.000000
iterations: 79638
3.505 seconds
```



Extra

While working on the Heuristic function, I realized that Manhattan distance using coordinates covers most of the orientation problems i.e., to move the corner into the right position with the right orientation, Manhattan distance estimates the correct number of moves in most case. The only case where Manhattan distance underestimates is when the corner is already in the correct position but not in the right orientation. I thought of accounting that but couldn't figure out a solution.