Lab 13

Solving the Rubik's cube, Part I

The set of all possible "moves" of the Rubik's cube is a good example of a group. Each possible move can be constructed as a product of moves consisting of rotating a single face of the Rubik's cube through a 90° rotation, either clockwise or counterclockwise. We will use the following notation for these moves:

One face of the Rubik's cube will be specified as the "Up" (top) face, and another as the "Front" face. These specifications will fix a "Down" (bottom) face (the face opposite the top face), a "Back" face (opposite the front face), and a "Right" and "Left" face (on the right or left as you look at the front face, with the top face on the top). We will use the following letters to denote the different faces:

U	Top
F	Front
R	Right
L	Left
B	Back
D	Down

Representing a turn of a face will be indicated by giving the letter of the face that we wish to turn, in upper case for a clockwise turn (as we look directly at the face), and in lower case for a counterclockwise turn. The turns will be listed in order from left to right in the order in which we perform them. For instance, the sequence `RFrf` would correspond to the following:

1. Turn the right face 90 degrees clockwise.
2. Turn the front face 90 degrees clockwise.
3. Turn the right face 90 degrees counterclockwise.
4. Turn the front face 90 degrees counterclockwise.

To keep the programming easier, we will not allow ourselves to turn the entire cube (in particular the squares in the center of each face will never change position). All moves that we allow will be moves of individual faces.

We will number the smaller faces of the Rubik's cube as indicated on the handout at

https://math.byu.edu/~doud/Math495R/Rubik-handout.pdf

With this numbering, you should check that turning the right face 90 degrees clockwise (the move denoted by `R`) would move the small face in spot 2 to spot 18, the face in spot 18 to spot 23, the face in spot 23 to spot 7, and the face in spot 7 to spot 2. The move `R` will actually involve five such 4-cycles: we have that as a permutation of the faces,

$$\texttt{R} = (2, 18, 23, 7)(3, 10, 22, 15)(6, 14, 19, 11)(26, 37, 46, 35)(30, 34, 42, 38)$$

We will define a variable `cube` that keeps track of the position of each sub-face as we manipulate the cube. When the cube is solved, `cube` will be a list of length 49, with the ith entry equal to i (note that there are only 48 subfaces that can move, but in order to keep the numbering simple, we will leave the zero entry always equal to 0).

1. Find the cycle structures for the moves `R`, `L`, `U`, `D`, `F`, and `B`. As an example, the cycle representing `R` would be

   `[[2,18,23,7],[3,10,22,15],[6,14,19,11],[26,37,46,35],[30,34,42,38]]`.

   Store these cycle structures in variables named `Right`, `Left`, `Up`, `Down`, `Front`, and `Back`. I suggest you crowdsource this part of the assignment.

2. Write a function `apply(move,cube)` that takes a list representing a movement of the cube (written as a product of cycles as in problem 1) and a list representing the cube (as described above), and performs the indicated move. It should actually change the values of the list `cube`, so that it does not actually need to return a value.

3. Write a function `execute(moves,cube)` that takes a string containing moves of the cube, and a position of the cube, and executes those moves on the cube.

   The function should apply all of the indicated moves to the cube, actually changing the values of the list. For instance, for a capital letter `R`, it should apply the permutation `Right` that you found above. For a lowercase `r`, it should apply the permutation `Right` three times (since a 90° counterclockwise move is the same as three 90° clockwise moves). After applying all moves specified in the string, the function should then return the sequence of moves that it made. If `moves` contains characters other than "`FBRLUDfbrlud`", these characters should be ignored, and not included in the returned variable.

   After this function is written and debugged, you should not change `cube` other than through this function (except to debug your code).

   In order to test the `execute` function, use it to find the orders of the following sequences of moves: `'RF'` should have order 105 (i.e., applying the move `RF` 105 times should return the cube to its initial position, but fewer than 105 times should not), `Ur` should have order 63, `rdL` should have order 180, and `RDFLBUrdflbu` should have order 360. Note that this last sequence of moves uses all twelve possible moves, so it should be a good test for your function.

4. Write functions: `align25(cube)`, `align26(cube)`, `align27(cube)`, `align28(cube)`, that put the subfaces numbered 25, 26, 27, and 28 in their correct positions. Each one should perform its task without changing the positions of the other subfaces in the set $\{25, 26, 27, 28\}$. Each function should return the sequence of moves that it makes.

   Here is my suggestion for how to write these functions.

   For `align25`, create a list of 24 sequences of moves. Entry 0 of this list would be the sequence of moves necessary to move subface 25 from position 25 to position 25; entry 1 of this list would be the sequence of moves necessary to move subface 25 from position 26 to position 25, and so on. Find the current position of subface 25, lookup the correct sequence of moves in the list, and execute the moves.

   For `align25`, here is what my function looks like:

```
def align25(cube):
    i=cube.index(25)
    move=['25','u26','uu27','U28','BLU29','RB30','uRB31','lb32',...][i-25]
    return execute(move,cube)
```

   In order to help me keep track of which entry of the list is which, I have included the starting position of subface 25 (so, for instance, "`Bru41`" would indicate the sequence of moves needed to move the subface in position 41 to position 25). If the `execute` function is correctly written, these extra digits will be ignored when the move is executed.

   The functions for `align26`, `align27`, `align28` can be constructed similarly.

5. Create four functions, `align1(cube)`, `align2(cube)`, `align3(cube)`, and `align4(cube)`, that return sequences of moves that put subfaces 1 through 4 in their correct positions without disturbing subfaces 25 to 28. These functions should work similarly to the ones written in part 4.

6. Create a function `solvetop(cube)` that has a scrambled cube as input, and returns a sequence of moves that solves the top level of the cube. The easiest way to do this would be to run each of the functions `align25(cube)`, `align26(cube)`, `align27(cube)`, `align28(cube)`, `align1(cube)`, `align2(cube)`, `align3(cube)`, and `align4(cube)` in the indicated order; put together their output into a single string, and return that string. Done efficiently, this can be a one or two line function.

You should test your `solvetop` program using some of the 100 scrambled cubes found:

math.byu.edu/∼doud/Math495R/Cubes.txt

If your routines work properly, subfaces 1–12 and 24–32 should all be put into the correct positions by `solvetop(cube)`.

If you need to see a graphical simulation of a cube, you can find one online at

math.byu.edu/∼doud/RubiksCube/

Enter a sequence of moves into the textbox at the top of the page, click the button labeled "Turn," and the program will execute the specified moves.

Lab 15

Solving the Rubik's cube, Part II

In this lab, we will continue programming our solution to the Rubik's cube. We will also construct several functions to help us debug our program.

Before beginning this lab, you should be sure that your functions from Lab 13 work correctly. They will be needed for this lab.

We will do the following in this lab: putting subfaces 33–40 in the correct position, and getting the edge faces 45–48 all on the bottom face. In the next (and final) Rubik's cube lab, we will complete the solution of the cube.

1. As a preliminary, we will create a function which scrambles a Rubik's cube. The function should have the following syntax: `scramble(cube,n=50)`. As input, it will take a vector representing a state of the Rubik's cube, and an (optional) integer n. It will then execute n random moves on the cube to scramble it. It should move the elements of `cube`, and return only the list of moves that it made.

Note that there are 12 possible moves: `"RLUDFBrludfb"`. To choose one at random use "`import random`" at the beginning of your program, and then `random.randint(0,11)` will choose a random integer from 0 to 11 (inclusive).

2. We now create a function that will translate sequences of moves designed to move subfaces on the front of the cube so that they move subfaces on other sides of the cube.

For instance, if we have a sequence of moves that takes the contents of the bottom front edge (location 43) and moves it to the right front edge (location 35), we want to be able to adjust it to move location 42 to 34 (on the right side), location 41 to 33 (on the back), or location 44 to 36, on the left.

To do this, hold up your cube handout, with the "Up" face on top, and the "Front" face facing you. Now rotate the cube 90° counterclockwise so that the "Left" face faces you. The face on the right hand side is now the face labeled "Front", so if you were to do an `R` move without reference to the labels on the cube, it would actually be a `F` move. Similarly, `L` would actually perform `B`, `F` would actually perform `L`, and `B` would actually perform `R`. The moves `T` and `D` would have the same effects as if the cube had not been rotated.

Note that if we do a sequence of moves that transfer the contents of location 43 to location 35, then after the rotation it would rotate the contents of location 44 to location 36.

Construct a function `toprotate(moves)` that will translate a series of moves by 90°, as above: in other words, it will make the following substitutions:

$$\texttt{R} \to \texttt{F}, \quad \texttt{F} \to \texttt{L}, \quad \texttt{L} \to \texttt{B}, \quad \texttt{B} \to \texttt{R}$$

$$\texttt{r} \to \texttt{f}, \quad \texttt{f} \to \texttt{l}, \quad \texttt{l} \to \texttt{b}, \quad \texttt{b} \to \texttt{r}$$

Probably the easiest way to do this is with the `translate` method on strings. To illustrate this method, run the following lines of code:

```
word="Convert abcde to numbers."
translation=str.maketrans("abcde","12345")
print(word.translate(translation))
```

Notice that each a has been turned into a 1, each b into a 2, and so on. Use this technique to make the translation function described above. Once you have `toprotate` working properly, the command `toprotate("RFLBrflb")` should return "`FLBRflbr`".

3. Create a function `edgesfrombottom(cube)` that will (assuming that the top face of the cube is solved) look at which faces are in locations 41 to 44, and if any face numbered between 33 and 40 is in one of these four spots will place it in its proper location. It should exit when all of the subfaces in locations 41 to 44 have numbers higher than 40. It should initialize a variable `result=""`, and use this variable to record and return any moves that it executes.

The first step to placing a subface in its proper place is to rotate it to the proper side of the cube. Note that all the edges on the front of the cube have numbers congruent to 3 mod 4; all the ones on the right have numbers congruent to 2 mod 4, all the ones on the back have numbers congruent to 1 mod 4, and all the ones on the left have numbers congruent to 0 mod 4. So, if we find a face numbered 33–40 in one of the spots 41–44, it is on the correct side of the cube if its number is congruent mod 4 to the position that it is in (i.e. if `(i-cube[i])%4)==0`. If it is in the wrong position, we need to use a `d`, `dd`, or `D` move to put it on the correct side.

Once we have it on the correct side, we need to move it to the correct position in the middle layer of the cube.

To do this, note that the sequence of moves `"drDRDFdf"` will move the contents of face 43 (the bottom edge of the front) to face 35 (the right edge of the front) without disturbing the top face or any of faces 33–40 except for 35 and 38. The sequence of moves `"DLdldfDF"` will move the contents of face 43 to face 39 (the left edge of the front) under similar restrictions.

These moves adjust the "Front" face of the cube. If we apply `toprotate` to them, they will have the desired effect on the "Left" face of the cube; namely moving the contents of position 44 to either position 36 or 40. Applying `toprotate` twice will affect the "Back" face of the cube, and applying `toprotate` three times will affect the "Right" face of the cube. (Try to find a formula for how many times you need to apply `toprotate`, instead of using multiple if statements.)

After checking each of locations 41 to 44, if no moves were executed (i.e., if `result` is still empty), the function should return the empty string. If any moves were executed, then we need to run the code again (since we may have disturbed a location that we had previously checked). Do this by returning `result+edgesfrombottom(cube)`.

In writing the code for this lab, the functions on the next page that test your code may be useful. If `testall(10000)` yields a positive result, then your code has worked correctly on 10000 scrambled cubes. These functions can be downloaded at

math.byu.edu/∼doud/Math495R/Lab16Debug.py

After you finish this lab, you should go to Lab 20 and start working on the next function.

To test (and debug) the function `edgesfrombottom`, you can use the following function:

```
def test(n):
   flag=True
   for i in range(n):                          #Tests your function on n cubes
      cube=list(range(49))                     #Initializes a solved cube
      t=scramble(cube)                         #Randomizes the cube
      t=solvetop(cube)                         #Solves the top face of the cube
      t=edgesfrombottom(cube)                  #Moves any side edges off of the bottom
      for position in range(41,45):            #Looks in locations 41 to 44
         if cube[position]<41:                 #If any side edges are there
            flag=False                         #Report a problem
            print("Your function has a problem")
   if flag:
      print("Your function seems to be fine.")
```

To test and debug the function `edgesfromsides`, use the following function.

```
def test2(n):
   flag=True
   for i in range(n):                          #Tests your function on n cubes
      cube=list(range(49))                     #Initializes a solved cube
      t=scramble(cube)                         #Randomizes the cube
      t=solvetop(cube)                         #Solves the top face of the cube
      t=edgesfrombottom(cube)                  #Moves any side edges off of the bottom
      t=edgesfromsides(cube)                   #Corrects the position of side edges
      if cube[33:41]!=list(range(33,41)):      #Checks the side edges
            flag=False
            print("Your function has a problem")
            print(cube[33:41])
   if flag:
      print("Your functions seem to be fine.")
```

The following function tests all of the functions that we have written so far.

```
def testall(n):
    flag=True
    for i in range(n):                          #Tests your function on n cubes
        cube=list(range(49))                    #Initializes a solved cube
        t=scramble(cube)                        #Randomizes the cube
        t=solvetop(cube)                        #Solves the top face of the cube
        t=edgesfrombottom(cube)                 #Moves any side edges off of the bottom
        t=edgesfromsides(cube)                  #Puts side edges in correct spots
        t=flipbottomedges(cube)                 #Flips bottom edges to the bottom
        if cube[1:13]!=list(range(1,13)):       #Tests top face
            flag=False
        if cube[25:41]!=list(range(25,41)):     #Tests second layer
            flag=False
        if min(cube[45:])<45:                   #Tests edges on bottom face
            flag=False
    if flag:
        print("Your code seems to be working")
```

Lab 17

Solving the Rubik's cube, Part III

In this lab we will solve the remaining parts of the Rubik's cube.

1. Create a function `edgesfromsides(cube)`. It should look at the subfaces in positions 33 to 36, and if one of them is incorrect, it should move this face to the bottom edge, and then run `edgesfrombottom(cube)` to place the edge in its correct position. Once all four positions from 33 to 36 have been dealt with, the second layer of the cube should be solved.

   To move a face from a side edge to the bottom, use the basic move "`drDRDFdf`" (for position 35), with `toprotate` applied the appropriate number of times to make it apply to the other three positions.

   At this point, you have code that solves the top two layers of the Rubik's cube.

2. We will now work on putting the subfaces numbered 45 to 48 on the bottom face of the cube. Many instruction manuals for solving the cube call this "creating a yellow cross", since they typically have the bottom face be yellow, and when all four bottom edges are on the bottom, they form the shape of a cross.

   In order to do this, we will create a function called `flipbottomedges(cube)`. It should change the entries of cube, and return the sequence of moves that it executes.

   It turns out that there are only four possibilities for how the bottom faces can be positioned.

   (1) None of them are on the bottom. In this case, the sequence of moves "`FLDldfBRDrdRDrdb`" will flip them all to be on the bottom.

   (2) Two of them are on the bottom, opposite from each other. In this case, after rotating the bottom so that the two correct ones are in position 46 and 48, the sequence of moves "`FLDldf`" will leave all four bottom faces on the bottom.

   (3) Two of them are on the bottom, adjacent to each other. In this case, after rotating the bottom so that the two correct ones are in positions 45 and 48, the sequence of moves "`RDFdfr`" will leave all four bottom faces on the bottom.

   (4) All four of the bottom edges are already on the bottom; in this case, nothing needs to be done.

Count the number of subfaces in position 45–48 that are numbered above 45–48. If this number is 0, we are in case 1 above. If the number is 4, we are in case 4 above. Otherwise, we are in either case 2 or case 3, and we should perform a sequence of `D` moves until location 48 contains a bottom face and location 47 does not. Then, depending on which of positions 45 and 46 contain bottom faces, we perform the appropriate sequence of moves.

3. We will now write a function `positionbottomedges(cube)` that will put the bottom edges in the correct positions. It should change `cube`, and return the sequence of moves that it executes.

   At this point, all four bottom edges are on the bottom of the cube, but they are probably not positioned correctly. It will always be possible to rotate the bottom face in such a way that either 2 or 4 of the bottom faces are positioned correctly. To do this, count the number of subfaces in positions 45–48 that are numbered correctly. As long as that number is less than two, execute a `D` move, and then repeat.

   If all four faces are positioned correctly, we are finished with this step. If only two faces are positioned correctly, we have more to do.

   If the two faces that are positioned correctly are opposite each other, they are either in positions 46 and 48 or positions 45 and 47. In the first case execute the sequence of moves: "`FDfDFDDfRDrDRDDrD`". In the second case, `toprotate` this sequence of moves once, and execute the result. This will put all four edge pieces in the correct positions.

   If the two faces that are positioned correctly are adjacent to each other, if they are in positions 45 and 48, execute the sequence of moves "`LDlDLDDlD`. If they are positioned differently, `toprotate` this sequence of commands the correct number of times to have the desired effect. This will put all four edge pieces in the correct positions. If the correct ones are in positions 45 and 46 a single `toprotate` should suffice; if they are in position 46 and 47, two `toprotate`s will be needed, if they are in positions 47 and 48, three `toprotate`s will be needed.

4. We will now write a function `positionbottomcorners(cube)` that will put the bottom corners in the correct positions. It should change `cube`, and return the sequence of moves that it executes. We will identify the bottom corner pieces by looking at positions 21, 22, 23, and 24. Note that if the face in one of these positions is congruent modulo 4 to its position, (i.e. if `(cube[i]-i)%4==0`) then it is in the correct position, but might be twisted incorrectly.

   To position the bottom corners, we have a basic move, "`rDLdRDld`" that fixes the corner containing position 24, and moves the other three corners in a clockwise pattern (twisting them as they move). If the cube has been solved to this point, it should be in one of the states below:

   (1) If none of the corners are in the correct position, execute the basic move. One will now be in the correct position.

(2) If one of the corners is correct, use toprotate on the basic move above the correct number of times so that it leaves the correct corner fixed. Then applying the toprotated basic move either once or twice will put all the corners in the correct position.

(3) It should never happen that two or three of the corners are in the correct position.

(4) If all four corners are in the correct position, we are done with this step.

5. We are now ready for the final step. This step will rotate each corner cube into the correct orientation.

   Write a function called `rotatebottomcorners(cube)`. It should change the cube, and return the sequence of moves that it makes.

   The function should consist of a loop that does the following four times.

   (1) While the face in position 21 is is numbered less than 21, execute the following move "`buBUbuBU`". (You may have to do this twice.)

   (2) Execute the move `D`.

6. Finally, we put together all of the functions that we have produced so far, to get a single function that solves the Rubik's cube. Your function might look somthing like the one below:

```
def solvecube(cube):
    result=""
    result+=solvetop(cube)
    result+=edgesfrombottom(cube)
    result+=edgesfromsides(cube)
    result+=flipbottomedges(cube)
    result+=positionbottomedges(cube)
    result+=positionbottomcorners(cube)
    result+=rotatebottomcorners(cube)
    return result
```

Once you have written this function, use the functions at the website

math.byu.edu/∼doud/Math495R/Lab16Debug.py

to test it on a large number of cubes.