MATH 495R, HOMEWORK 16
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 14 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 the 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:

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

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

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)`.

4. 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.

5. 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.

Find a way to identify the pattern of bottom edges, and apply the appropriate moves.

Our code now solves the top two layers of the Rubik's cube, and orients the bottom edges correctly (although they may not yet be in the correct positions).

In the next lab, we will write functions that do the following:

(1) Position the bottom edges correctly.
(2) Position the bottom corners correctly.
(3) Orient the bottom corners correctly.

At that point, our code will be able to completely solve the Rubik's cube.

In writing the code for this lab, the functions on the last 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

`https://math.byu.edu/~doud/Math495R/Lab16Debug.py`

.

Part 5 will be due with the next lab!

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")
```