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.

Before beginning this lab, you should be sure that your functions from Labs 14 and 16 work correctly. They will be needed for this lab.

We will do the following in this lab: putting subfaces 45–48 in the correct position, position the bottom corners pieces correctly, and then orient the bottom corner pieces correctly.

1. (Note that this portion of the lab was originally part of lab 16).

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.

2. 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 more than 44. 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.

3. 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.

4. 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`.

5. 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 following function to test in on a large number of cubes.

```
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+=solvecube(cube)                      #Solves the cube
        if cube!=list(range(49)):               #Checks that the cube is solved
            flag=False
        testcube=list(range(49));
        u=execute(t,testcube)                   #Checks that the moves actually
        if testcube!=list(range(49)):           #   solve the cube.
            flag=False
    if flag:
        print("Your code seems to be working")
```