7. Rubik’s Cube

12/09/2019

INTRODUCTION

Rubik's Cube is a puzzle cube, and the world's biggest selling toy of all time with over 300,000,000 (300 million) sold. It was invented in 1974 by
Hungarian sculptor and professor of architecture Ernő Rubik. In a classic Rubik's Cube, each of the six faces is covered by nine stickers, each of one of six solid colours: white, red, blue, orange, green, and yellow. In currently sold models, white is opposite yellow, blue is opposite green, orange is opposite red, and some might replace blue with black or purple.

PYTHON IMPLEMENTATION

```
DIRECTIONS = (
    ( 1,  0,  0), # right
    ( 0,  1,  0), # up
    ( 0,  0,  1), # front
    (-1,  0,  0), # left
    ( 0, -1,  0), # down
    ( 0,  0, -1)  # back
)

DIRECTIONS_NAME = dict(zip(DIRECTIONS, "rufldb"))

def cross(axis,direction):
    # cross product
    return (axis[1]*direction[2] - axis[2]*direction[1],
             axis[2]*direction[0] - axis[0]*direction[2],
             axis[0]*direction[1] - axis[1]*direction[0])

def dot(va,vb):
    # dot product
    return sum(a*b for (a,b) in zip(va,vb))

def scale(alpha, v):
    # scaling a vector
    (x,y,z) = v
    return (alpha*x, alpha*y, alpha*z)

def add(u,v):
    # adding two vectors
    return (u[0] + v[0], u[1] + v[1], u[2] + v[2])
```

```python
def rotate(axis, u):
    # rotation by a quarter in the
    # positive sense around a normal vector.
    axis_projection = scale(dot(axis,u), axis)
    ortho_projection = cross(axis, u)
    return add(axis_projection, ortho_projection)

def degree(coords):
    # Given the position of a block
    # return the number of faces that
    # that are visible.
    return sum(map(abs,coords))

class NonOrientedCube(object):
    def rotate(self, axis):
        return self

class OrientedCube(object):
    __slots__=("orientation")
    def __init__(self, orientation=DIRECTIONS[:2]):
        self.orientation = orientation
    def rotate(self, axis):
        return OrientedCube(
            tuple(rotate(axis,u)
            for u in self.orientation)
        )
    def __eq__(self, other):
        return self.orientation == other.orientation
    def __ne__(self, other):
        return self.orientation != other.orientation

zero_oriented = {
    coords: OrientedCube()
    for coords in product(*([(-1,0,1)]*3))
    if degree(coords) >= 2
}

zero_non_oriented = {
    coords: NonOrientedCube()
    for coords in product(*([(-1,0,1)]*3))
    if degree(coords) >= 2
}

def turn(axis, rubix_cube):
    parts = {}
    for (coord, cube) in rubix_cube.items():
        if any(x == y != 0 for x, y in zip(axis, coord)):
            # this cube is on the face rotating,
            # let's rotate it and register it to
            # its destination.
            new_cube = cube.rotate(axis)
            new_coord = rotate(axis, coord)
            parts[new_coord] = new_cube
        else:
            # this cube is not on the face that is rotating.
            parts[coord] = cube
    return parts

def project(rubix, d):
    return {
        coords: v
        for (coords, v) in rubix.items()
        if degree(coords) == d
    }

def sides(rubix):
    return project(rubix, 2)

def corners(rubix):
    return project(rubix, 3)

OPERATIONS = [
    [ direction ]*i
    for direction in DIRECTIONS
    for i in range(1,4)
]

def sequence(seq, rubix):
    for axis in seq:
        rubix = turn(axis, rubix)
    return rubix

def differences(rubix_1, rubix_2):
    return [
        k
        for k in rubix_1.keys()
        if rubix_1[k] != rubix_2[k]
    ]

def browse_with_length(els, n):
    if n==0:
        yield []
    else:
        for head in els:
            for tail in browse_with_length(els, n-1):
                yield head + tail

def browse_tuples(els):
    for n in count(1):
        for seq in browse_with_length(els, n):
            yield seq

def all_on_one_face(positions):
    for els in zip(*positions):
        if len(set(els)) == 1:
            return True
    return False

def search_orbit(seq, fixed_rubix, diff_rubix, max_depth):
    iter_fixed_rubix = fixed_rubix
    iter_diff_rubix = diff_rubix
    for i in range(1,max_depth+1):
        iter_fixed_rubix = sequence(seq, iter_fixed_rubix)
        iter_diff_rubix = sequence(seq, iter_diff_rubix)
        if not differences(fixed_rubix, iter_fixed_rubix):
            diff = differences(diff_rubix, iter_diff_rubix)
            if not diff:
                break # we ran through a full orbit.
            elif all_on_one_face(diff) and len(diff) <= 3:
                return (seq, i, diff)

DIRECTIONS_NAME = dict(zip(DIRECTIONS,
    ["right", "up", "front", "left", "down", "back" ]))

def operation_to_string(seq):
    return "-".join([ DIRECTIONS_NAME[axis]
        for axis in seq ])

def search_step2_move():
    for seq in browse_tuples(OPERATIONS):
        seq = [DIRECTIONS[0]] + seq
        if len(seq) % 2 == 0:
            magic_move = search_orbit(seq,
                sides(zero_oriented),
                corners(zero_non_oriented), 4)
            if magic_move:
                (operation, repeat, dist)=magic_move
                print operation_to_string(operation),
                print "x" +str(repeat),
                print dist
                break

search_step2_move()
```

```
def search_step3_move():
    for seq in browse_tuples(OPERATIONS):
        seq = [DIRECTIONS[0]] + seq
        if len(seq) %2 == 0:
            corners_non_oriented = dict(
                zero_oriented,
                **corners(zero_non_oriented))
            magic_move = search_orbit(seq,
                corners_non_oriented,
                corners(zero_oriented),
                6)
            if magic_move:
                (operation, repeat, dist)=magic_move
                print operation_to_string(operation),
                print "x" +str(repeat),
                print dist
                break

search_step3_move()
```

CONCLUSION

All the difficulty left here is to find a magic move which leaves sides untouched and yet have some effect on the corners. Let's call such a combination a magic move!

When trying to find a magic move, especially without a computer, a good trick is to test many moves and consider for each of them what happens if you repeat this moves over and over. The images obtained by repeating the operation is also called an orbit.

When doing that by hand, it can be tested very rapidly by representing the underlying permutations as a union of cycles.

But with a computer we can do all this very simply.