Jason Wong
CMSC 404
Assignment 3 Report

Computing a table is done by using a breadth-first search, implemented by adding Nodes that consist of a String representing the rotations needed to get to a state, and a corresponding Integer depth value, to an ArrayList. At the start of the search, a Node is constructed with an empty string and a depth value of one. The search then removes the Node at the first index, constructs a solved RubiksCube, applies the necessary rotations, and calls the methods to get the relevant cubies from the cube as a String with each character representing a face on the cubie, which are then encoded. The different cubie types are stored in their own array, with the key equal to the encoding, and then the values are saved in a file sequentially, with the keys equal to the offsets of the values in the file.

The corner cubies are encoded with the summation $i! * x * 3^7 + 3^{i-1} * y$ for i=7 down to i=1. X is equal to the index of the cubie in one of two ArrayList<String>s that have the possible sequences of colors a cubie can have at a certain corner. The array to choose from is determined by the position of the cubie relative to the entire cube. If a corner cubie's sequence of colors is found in one array, all adjacent corner cubies will have their sequence of colors found in the other array. (This is due to polling the faces of a cubie in a predetermined order; one 90 degree rotation in any direction causes two of the colors in the sequence to swap places). As a cubie is found, the element at index i is removed from both of the ArrayLists before the next iteration of the summation. Y is equal to the number of rotations that must be made to the sequence of colors in order to match the cubie's actual order (i.e. base sequence YGR has Y=0, one rotation produces GRY with Y=1, and two rotations produce RYG with Y=2).'

Both pairs of edge cubies are encoded with the summation $i!/6! * x * 2^6 + 2^{i-6} * y$ for i=11 down to i=6. Since only six cubies need to be encoded, i! is divided by 6!, leaving six remaining factors to multiply by while preserving the one-to-one mapping of the encoding. X and Y are determined like corner cubies are, with one difference: X pulls from one array of all twelve side cubies.

As a result of the encoding function, the corner cubie table has 88179840 indices, while the edge cubie tables have 42577984 indices. I only had resources to compute the tables up to depth five, making the number of non-zero entries in the tables 236,552 for the corners, 276,542 for six edges, and 276,526 for the other six edges. When the tables are read into the solver program, they are stored in char arrays, making the total memory usage of the tables 346 megabytes.

The program can be run with `java SolveCube.java filepath`

Sample State Output

cube0	Y3R3B2W3R3W2G3O2W1
cube1	no solution after 10 minutes
cube2	no solution after 10 minutes
cube3	no solution after 10 minutes
cube4	no solution after 10 minutes
cube5	no solution after 10 minutes
cube6	no solution after 10 minutes
cube7	no solution after 10 minutes
cube8	no solution after 10 minutes
cube9	no solution after 10 minutes