Table of Contents

CAB 301 FINAL PROJECT REPORT

Student: Duy Pham (Daniel)

ID: n10640754

1. Introductions

This report delivers a solution which will effectively solve any cases of a scrambled "**2 × 2 Rubick's cube**" – "**effectively**" means smallest number of moves made to solve a case of a scrambled Rubick. Giving a knowledge that whenever the user uses legal moves on the cube it can change into another different form of itself. By using these legal moves on the Rubick we can find the "cube's configuration" or in another hand, we can find moves that turn a scrambled cube into solved cube. Each cube's configuration can link with each other by a legal move, so it is a point-to-point relation.

Example of Rubick's gameplay flow:

Solved cube -----legal move 1---- cube's config 1.

cube's config 1 -----legal move 2---- cube's config 2

cube's config 2 -----legal move 3---- cube's config 3

cube's config 3 -----legal move 4---- cube's config 4

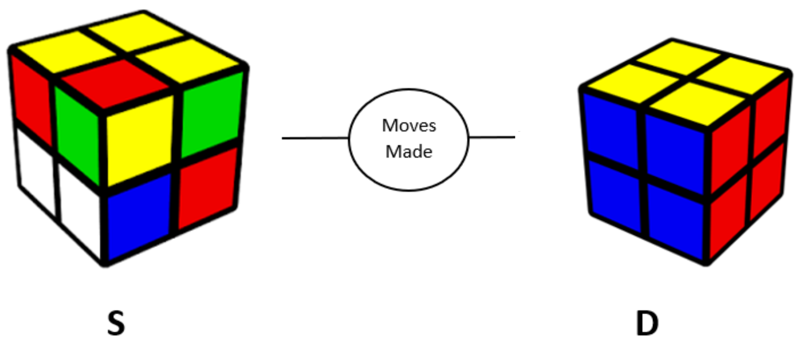
.....

Cube's config "n" -----legal move "n" ----- scrambled cube

The increment of the config number notation will be the depth of the problem. For example, cube's config 1 means the player makes one move and this one move makes the cube's depth problem increase 1.

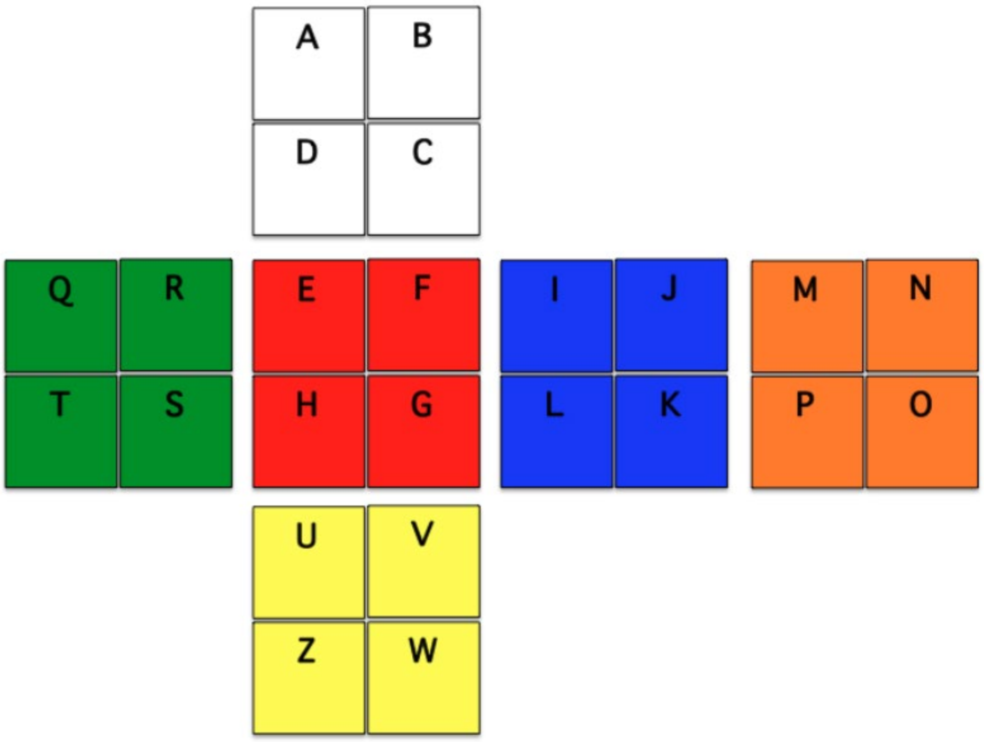
2. Instance Model

As follow the introduction, the program will take a scrambled Rubick's cube as an instance to start solving. By encoding this instance into graph theory, we will have **vertex "S"** as the **unsolved Rubick's cube** in the first start, and **vertex "D"** will be the situation when **Rubick's cube got solved.** (It will take "n" moves to get there)



Before the cube can be solved the instance must be **valid** in the first place, but a scrambled Rubick's cube is come from the solved one (match squares' colour in each surface). Therefore, to encode an instance for this program we need to encode the Cube in its' solved condition and from that provide a suitable instance unless will throw error of a never-ending task.

Below is an encoding of a solved Rubick's cube: (this will be an encode of the "**cube's config**" **in section 1**)



In clockwise order from top left sticker: (number next to character is index)

Up layer = { A0, B1, C2, D3}
Front layer = { E4, F5, G6, H7}
Right layer = { I8, J9, K10, L11}
Back layer = { M12, N13, O14, P15}
Left layer = { Q16, R17, S18, T19}
Down layer = { U20, V21, W22, Z23}

Turn above sets into a string: (*We should use a set of colours notation because it got the same value for each face, therefore will be better for coding*)

Cube config = {A, B, C, D, **OR** **Cube config = {W, W, W, W**

E, F, G, H, **R, R, R, R,**

I, J, K, L, **B, B, B, B,**

M, N, O, P, **O, O, O, O,**

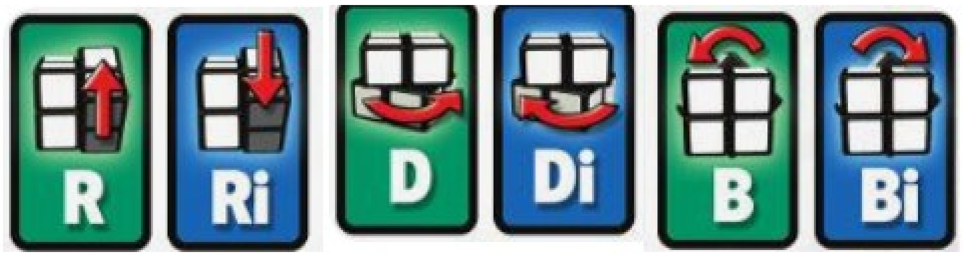
Q, R, S, T, **G, G, G, G,**

U, V, W, Z} **Y, Y, Y, Y}**

To create a legal instance, it needs legal moves.

There are 2 types of move for the Rubick (90degree clockwise spin and 90degree counterclockwise spin). For each moves the cube "depth" will be deeper (Depth – means the distance or how far that we transform the cube's configuration).

Possible moves: (Note: I can reduce half of the cube's moves because we don't care about its' orientation, moves of a surface will be similar to its' opposite surface's moves, for example bottom and top will have same twists)



Therefore, the model of the instance will be a Set of 24 characters of colour notations. In the order:

UP LAYER -> FRONT LAYER -> RIGHT LAYER -> BACK LAYER -> LEFT LAYER -> DOWN LAYER

Config *= {W,W,W,W, R,R,R,R, B,B,B,B O,O,O,O, G,G,G,G, Y,Y,Y,Y}*

Example: (encode the test data)

***Config**= {G,G,W,O   Y,G,O,Y,   R,Y,W,G,   R,W,W,O   O,B,B,B,   R,Y,B,R}*

3. Solution Model

To solve the problem, we will need to identify which cube config(vertices) links to which config through a legal move (edge), and in what order. To do that, program will keep linking the cube config together (this iteration will create a graph form because it needs to check 6 possibilities in each cube config) until it can link "s" (the root) vertices with the "d" vertices. Therefore, the solution can be a sequence of the cube's transformation ***(s, cube config 1, cube config 2, cube config 3, cube config 4, …., cube config n-1, d)*** *thus a sequence of legal moves* ***(legal move 1, legal move 2, legal move 3, … n)****. Therefore,* **depth of the Rubick problem** *(how many times the cube got scrambled) will be how many moves it needed to get solved. Furthermore, the cube is solved when it got all cubies in each faces got the same colour (ignore the perspective of the cube => the encode version of solved cube will not be in order of faces, to be exact there are 24 cases of solved cube)*

4. Problem Model

Now we will model the problem. As said before the problem of the Rubick is a graph (undirected because moves can be reversible), therefore, it can be encoded in an undirected and connected graph format G = (V, E), with V is the set of vertices of G, which are the cube's configurations linked together through a legal move. Assume "L" is a set of cube config pairs, which has a legal move between them.

$$V = \{\, a : (a,b) \in L\} \cup \{\, b : (a,b) \in L\}$$

And E (set of edges of graph G) will be the legal move between two cube configs.

$$E = L \cup \{(a,b) : (b,a) \in L\}$$

Therefore, existed a path of scrambled Rubick "s" to solved Rubick "d" in graph G (path of in order vertices). Mentioned that there are paths that could create loop inside the graph (example: keep using a single move), therefore it cannot be a tree form.

5. Solution Method

A method of using breadth first search will be applied to solve this problem, Because the problem is to find the shortest path between vertices, therefore we will need to check through each possibility in each depth of the problem (each depth will have 6 cases- represented as 6 moves and will keep rising in powers of 6). We have the degree of a set of vertices in depth(distance) **n**.

$$d(D_n) = 6^n \quad \text{(6 moves lead into 6 cube states)}$$

Base case: Assume that we found the cube wanted state in the first case – vertices 0 "v_0" and $d(D_0) = 6^0 = 1$ (only one cube config in the graph). The distance or level of the problem right now is:

$$D_0 = \text{depth}[v_0] = 0$$

(Moves needed to solve the Rubick)

For recursive case – assume the vertices v_n in depth(level) n, which will have the distance (D_n – distance from start vertices to destination) $v_n = d$ (destination) assigned in the level.

$$D_n = \text{depth}[v_n] = n \text{ (moves)}$$

Therefore, for every vertex v, fewest edges to get from s to v will be the level of an expanding graph which contains that vertex:

$$\begin{cases} depth[v_n] \; if \; v \; is \; assigned \; in \; the \; level \\ infinite \qquad\qquad else \; (no \; path) \end{cases}$$

I created a list of sets called D this list will store all $D_0, D_1, D_2, \dots D_n$ the Length of the list will be the number of moves.

For short, to find shortest path of a vertex v (fewest move):

+ take s.	
+ Descendant[s]	(Depth 1)
+ Descendant[Descendant[s]]	(Depth 2)
+ ….	(Depth n)
+ until v (or none)	(Depth of destination)
	= number of moves