On to Rubik's Cube

```
n = 54;
g1 = Cycles[{{1, 18, 45, 28}, {2, 27, 44, 19}, {3, 36, 43, 10}, {46, 52, 54, 48}, {47, 49, 53, 51}}];
g2 = Cycles[{{7, 16, 39, 30}, {8, 25, 38, 21}, {9, 34, 37, 12}, {13, 15, 33, 31}, {14, 24, 32, 22}}];
g3 = Cycles[{{28, 31, 34, 48}, {29, 32, 35, 47}, {30, 33, 36, 46}, {37, 39, 45, 43}, {38, 42, 44, 40}}];
g4 = Cycles[{{1, 3, 9, 7}, {2, 6, 8, 4}, {10, 54, 16, 13}, {11, 53, 17, 14}, {12, 52, 18, 15}}];
g5 = Cycles[{{1, 13, 37, 46}, {4, 22, 40, 49}, {7, 31, 43, 52}, {10, 12, 30, 28}, {11, 21, 29, 19}}];
g6 = Cycles[{{3, 48, 39, 15}, {6, 51, 42, 24}, {9, 54, 45, 33}, {16, 18, 36, 34}, {17, 27, 35, 25}}];


(* Apply tau, then sigma, so smallcircle works like regular composition of functions*)

σ_ ∘ τ_ := PermutationProduct[τ, σ]


(* Here is the code for filling the table. The variable gee counts the
 length of the factorization of the current permutation in terms of the g's,
and sig keeps track of how the table entries are obtained from one another;
gwiz and sigz are temporary versions of these variables, respectively. Finally,
count keeps track of the number of times a slot in the table is reassigned.*)

Remove[σ]; Remove[sig]; Remove[gee]; listsig = {};
σ_ ∘ τ_ := PermutationProduct[τ, σ];
geed[{i_, j_}] := gee[{i, j, count[i, j]}];
Feed[Cycles[{}], _, _] := 1 + 1;
Feed[τ_, sigz_, gwiz_] := Module[{i, j, k, l, r},
   i = Min[PermutationSupport[τ]];
   j = PermutationReplace[i, τ];
   If[Head[σi,j,count[i,j]] === Cycles, If[gee[{i, j, count[i, j]}] > gwiz, count[i, j]++;
     σi,j,count[i,j] = τ;
     gee[{i, j, count[i, j]}] = gwiz;
     sig[{i, j, count[i, j]}] = sigz;
     Feed[InversePermutation[σi,j,count[i,j]] ∘ σi,j,count[i,j]-1,
      Append[Reverse[sigz^(-1)], {i, j, count[i, j] - 1}], gwiz + gee[{i, j, count[i, j] - 1}]],
     Feed[InversePermutation[σi,j,count[i,j]] ∘ τ, Prepend[sigz, {i, j, count[i, j]}^-1],
      gwiz + gee[{i, j, count[i, j]}]]],
    (*Else*) count[i, j] = 1;
    σi,j,count[i,j] = τ;
    gee[{i, j, count[i, j]}] = gwiz;
    sig[{i, j, count[i, j]}] = sigz;
    listsig = Append[listsig, {i, j}];
    (*Add code to sort list of defined sigmas according to
     length of their g decomposition. This is done to avoid too much doubling back*)
    listsig = SortBy[listsig, geed];
    For[r = 1, r ≤ Length[listsig], ++r, k = listsig[[r]][[1]]; l = listsig[[r]][[2]];
     Feed[σi,j,count[i,j] ∘ σk,l,count[k,l], Append[{{i, j, count[i, j]}}, {k, l, count[k, l]}],
      gee[{i, j, count[i, j]}] + gee[{k, l, count[k, l]}]]; Feed[σk,l,count[k,l] ∘ σi,j,count[i,j], Append[
       {{k, l, count[k, l]}}, {i, j, count[i, j]}], gee[{i, j, count[i, j]}] + gee[{k, l, count[k, l]}]];
    ]
   ]];
$RecursionLimit = 10^6;

Do[Feed[gi, {g[i]}, 1], {i, 1, 5}]

(* Feed2 takes an element of the group, once the table has been filled,
and gives a factorization in terms of the table entries*)
```

```
(* Let's find the worst case for the number of moves in a factorization*)

geef[{i_, j_}] := Which[IntegerQ[geed[{i, j}]] == True, geed[{i, j}], IntegerQ[geed[{i, j}]] == False, 0]
worstcase = Sum[Max[Table[geef[{i, j}], {j, i, n}]], {i, 1, n}]
```

2463

```
(* That's not so bad*)


Feed2[Cycles[{}], _] := 1 + 1;
Feed2[τ_, ifact_] := Module[{i, j, k, l},
  i = Min[PermutationSupport[τ]];
  j = PermutationReplace[i, τ];
  fact = Append[ifact, {i, j, count[i, j]}];
  Feed2[InversePermutation[σ_{i,j,count[i,j]}] ∘ τ, fact];
 ]
$RecursionLimit = 10^6;

(* Siggy expands the table entires into their factorizations in terms of the group generators*)

siggy[{i_, j_, count_}] := Piecewise[
   {{Reverse[sig[{1/i, 1/j, 1/count}]^(-1)], Min[i, j] < 1}, {sig[{i, j, count}], Min[i, j] ≥ 1}}];
siggy[g[i_]] := g[i];
siggy[g[i_]^-1] := g[i]^-1;

rule = list_ :> Flatten[Map[siggy, list], 1] ;
relations = {{lft___, c_, c_, rght___} :> {lft, c^2, rght},
   {lft___, c_, c_^x_, rght___} :> {lft, c^(1+x), rght}, {lft___, c_^x_, c_, rght___} :> {lft, c^(1+x), rght},
   {lft___, c_^x_, c_^y_, rght___} :> {lft, c^(x+y), rght}, {lft___, 1, rght___} :> {lft, rght}, c_^x_ :> c^Mod[x,4]};

(* factorization gives a compact factorization in terms of the symbolic permutations, g[1],...,g[6]*)


factorization[τ_] := Module[{finalfact}, Feed2[τ, {}];
   finalfact = fact //. rule //. relations;
   finalfact];

(* replace is a set of rules to put the replace the symbolic permutations by the actual permutations*)

replace = {g[i_] :> g_i, g[i_]^2 :> g_i ∘ g_i, g[i_]^3 :> g_i ∘ (g_i ∘ g_i)};

(* longprod takes a product of a list of permutations of
 arbitrary length. With it we can check if our factorization is correct*)

longprod[listperm_] := Module[{templist, prod}, prod = Cycles[{}];
  ilength = Length[listperm];
  templist = listperm;
  Do[prod = (Last[templist] /. replace) ∘ prod;
   templist = Delete[templist, -1], {i, 1, ilength}];
  prod]
```

```
factorization[g6]
```

$$\{g[1]^3, g[3], g[1]^2, g[3]^3, g[1], g[3], g[1]^3, g[3]^2, g[4]^3, g[1]^2, g[3], g[1]^2, g[3]^2, g[1]^3, g[3], g[1]^2,$$
$$g[3]^3, g[1], g[3], g[1]^3, g[3]^2, g[4]^3, g[1]^2, g[3], g[1]^2, g[3]^2, g[1]^3, g[3], g[1]^2, g[3]^3, g[1],$$
$$g[2]^3, g[5], g[3], g[1], g[3]^3, g[1]^3, g[3], g[1]^2, g[3]^3, g[1], g[3]^2, g[1]^2, g[3]^3, g[1]^2, g[4],$$
$$g[3]^2, g[1], g[3]^3, g[1]^3, g[3], g[1]^2, g[3]^3, g[1], g[3]^2, g[1]^2, g[3]^3, g[1]^2, g[4], g[3]^2, g[1],$$
$$g[3]^3, g[1]^3, g[3], g[1]^2, g[3]^3, g[1], g[2]^3, g[1]^3, g[3], g[1], g[4]^3, g[5]^3, g[1]^3, g[3]^3, g[1]^3,$$
$$g[3]^3, g[1], g[2]^3, g[1]^3, g[3], g[1], g[3], g[1], g[2], g[3]^3, g[2]^2, g[4]^3, g[1], g[3]^3, g[1],$$
$$g[3], g[1]^3, g[3]^3, g[5]^3, g[2], g[1]^3, g[3], g[1]^3, g[4], g[1]^3, g[3]^3, g[1], g[2]^3, g[1]^3, g[3]^3,$$
$$g[1], g[4]^3, g[1]^3, g[3]^3, g[1], g[3]^2, g[1]^2, g[3]^3, g[1]^2, g[4], g[3]^2, g[1], g[3]^3, g[1]^3, g[3],$$
$$g[1]^3, g[4], g[1], g[3]^3, g[1], g[3], g[1]^3, g[5], g[4], g[3]^2, g[2]^3, g[1]^3, g[3]^3, g[1], g[2],$$
$$g[1]^3, g[3], g[1]^2, g[3]^3, g[1]^3, g[3], g[1]^3, g[2]^3, g[1], g[3]^3, g[1], g[3], g[1]^3, g[2], g[1],$$
$$g[3]^3, g[2]^3, g[1]^3, g[3], g[1]^2, g[2], g[1]^3, g[3], g[1]^2, g[3]^3, g[1]^3, g[3]^3, g[1], g[2]^3, g[1]^3,$$
$$g[3], g[1], g[3], g[1], g[2], g[3]^2, g[2]^2, g[3], g[1], g[3]^3, g[1], g[3], g[1]^3, g[3]^3, g[5]^3, g[2],$$
$$g[1]^3, g[3], g[1]^3, g[2]^2, g[1], g[3]^3, g[1], g[2]^2, g[1]^3, g[3], g[1], g[2], g[5], g[3], g[1], g[3]^3,$$
$$g[1]^3, g[3], g[1]^2, g[3]^2, g[1], g[2]^3, g[1]^3, g[3], g[1], g[2], g[1]^3, g[3]^2, g[1], g[2]^3, g[1],$$
$$g[3]^3, g[1]^3, g[3], g[1]^3, g[2], g[1], g[3]^3, g[1], g[3], g[1]^3, g[3]^2, g[1]^3, g[3]^3, g[1], g[3]^3,$$
$$g[1]^3, g[3], g[1], g[3], g[1]^3, g[3], g[1]^2, g[3]^3, g[1], g[3]^3, g[1]^3, g[3], g[1]^2, g[3], g[1]^2,$$
$$g[3]^3, g[1]^3, g[3]^3, g[1]^2, g[3], g[1]^2, g[3]^3, g[1]^3, g[3]^2, g[1], g[3]^3, g[1], g[3]^3, g[1]^2, g[3],$$
$$g[1]^3, g[3]^3, g[1], g[3]^2, g[1], g[2]^3, g[1]^3, g[3], g[1], g[2], g[1]^3, g[3]^2, g[1], g[2]^3, g[3],$$
$$g[2], g[1]^3, g[3], g[1], g[3], g[1]^3, g[3]^2, g[1], g[3]^2, g[1]^3, g[3]^2, g[1], g[3]^3, g[1]^3, g[3]^3,$$
$$g[1]^3, g[3]^2, g[1]^3, g[3]^2, g[1], g[3]^3, g[1], g[3]^3, g[1]^2, g[3], g[1]^3, g[3]^3, g[1]^3, g[3]^3, g[1],$$
$$g[3]^3, g[1]^3, g[3], g[1]^2, g[3], g[1]^2, g[3]^3, g[1]^3, g[3]^2, g[1], g[3]^2, g[1]^3, g[3], g[1]^2, g[3]\}$$

```
(* Read this factorization as if there were small
 circles between each entry. So g6 is equivalent to applying g3,
then g1 twice, then g3, and so on. To solve a cube given such a factorization,
we proceed from left to right, doing the inverse of everything stated*)
```

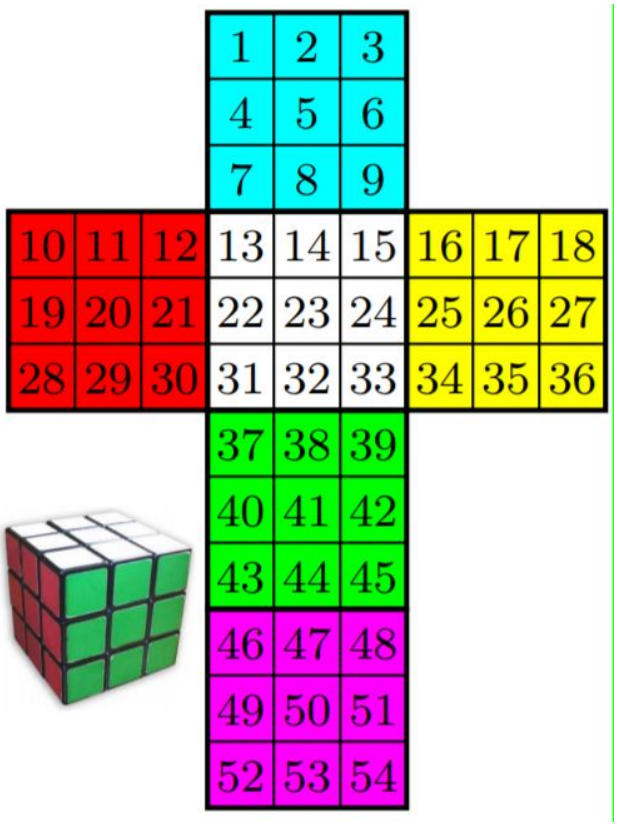
```
longprod[factorization[g6]] == g6
```

```
True
```

```
(* So, the above factorization is correct. This shows us how to
 accomplish a rotation of the yellow face by rotating the other 5 faces*)
```

```
(*Now write a program to take a permutation and put it into cycle notation. Once I have this,
I can begin solving real cubes*)
```

```
FuncToCycle[σ_, n_] := Module[{list, in, cyc, curcyc}, list = Range[n];
  in = 1;
  cyc = {{}};
  curcyc = {};
  While[Length[list] > 0, While[MemberQ[list, in], list = Delete[list, Position[list, in]];
    If[MemberQ[Flatten[cyc], in], Break[], curcyc = Append[curcyc, in]];
    If[in ≠ σ[in], in = σ[in]] ];
   If[Length[list] > 0, in = First[list]];
   cyc = Append[cyc, curcyc];
   curcyc = {}];
  Cycles[cyc]]
```




```
(* As a test, let us write down the rotation of the blue face as a function,
and take its cycle representation, and compare that to g4 above*)
Clear[f]
f[x_] = x;
f[1] = 3;
f[2] = 6;
f[3] = 9;
f[6] = 8;
f[9] = 7;
f[8] = 4;
f[7] = 1;
f[4] = 2;
f[13] = 10;
f[14] = 11;
f[15] = 12;
f[16] = 13;
f[17] = 14;
f[18] = 15;
f[54] = 16;
f[53] = 17;
f[52] = 18;
f[10] = 54;
f[11] = 53;
f[12] = 52;
```

```
FuncToCycle[f, n] == g4
```

```
True
```

```
(* So it works correctly. Hence, given a real cube,
if we enter the arrangements of the colors as a map f,
we can then determine the cycle representation for that permutation, and feed it to our solver.*)
```