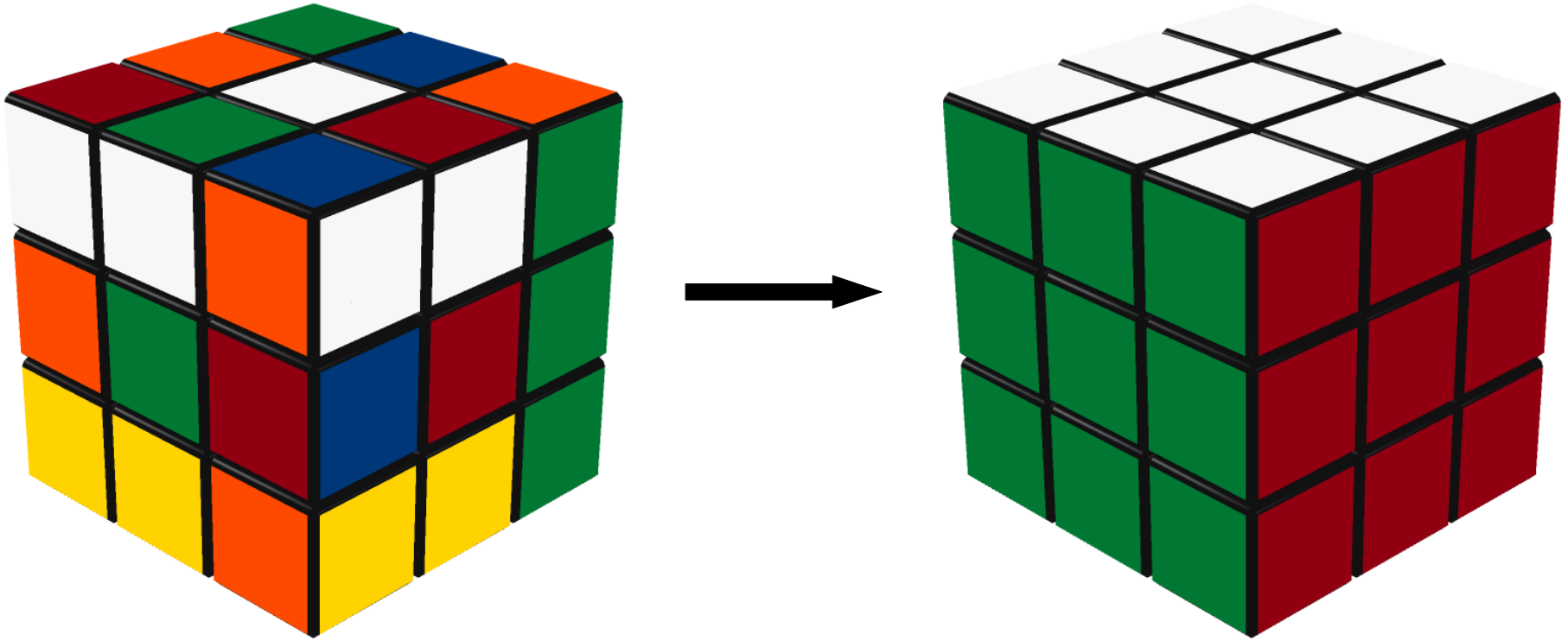
Soluções Eficientes para o Cubo Mágico

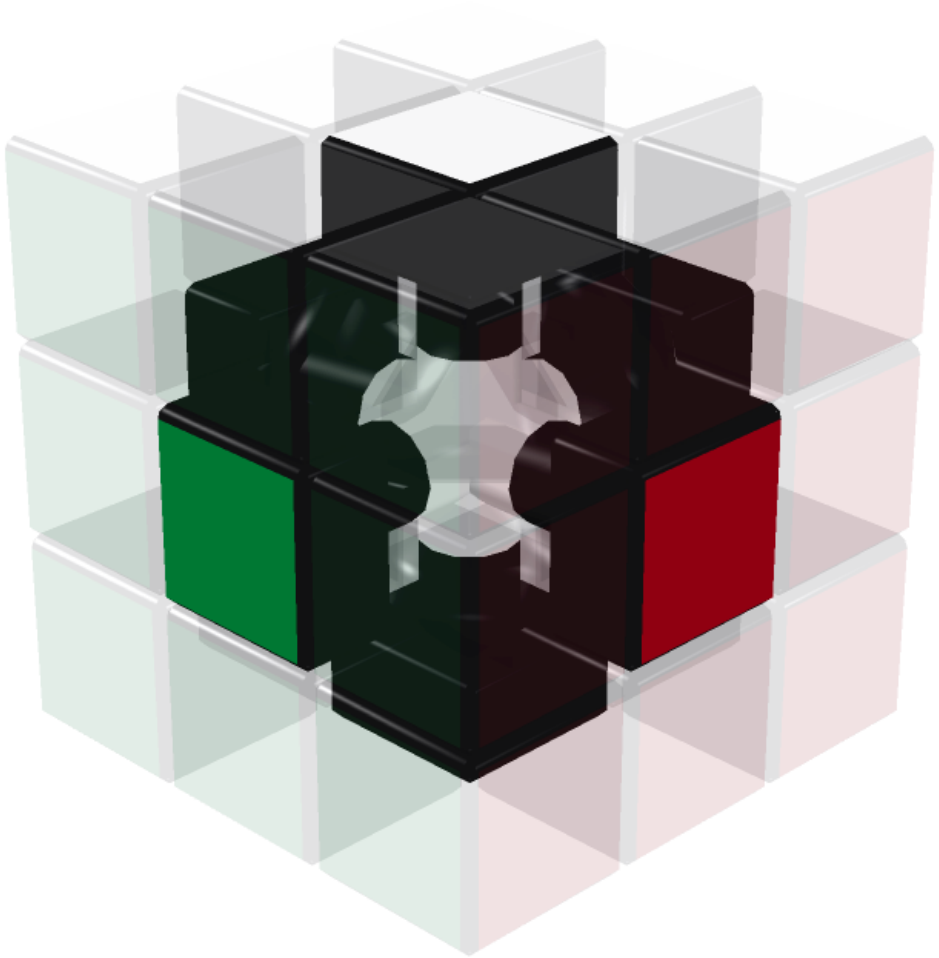
Walter Pereira Rodrigues de Souza

Orientadora Nina S. T. Hirata

Cubo Mágico

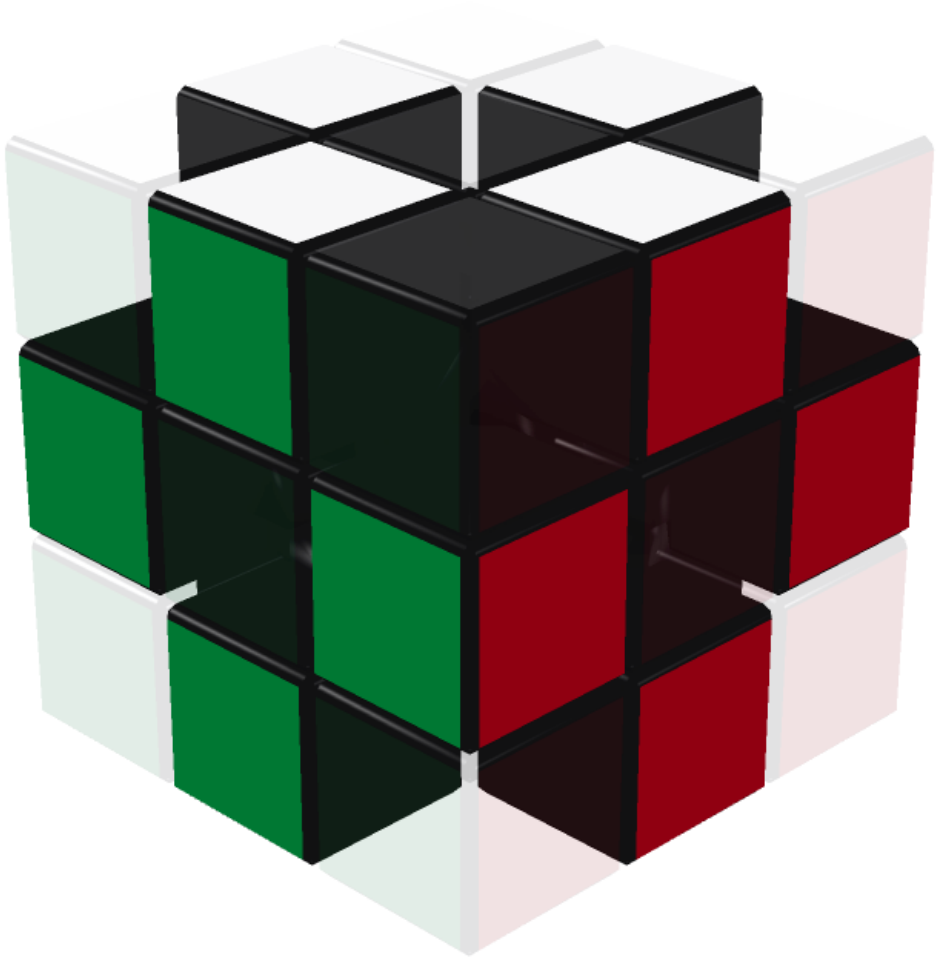


Mecanismo - Centros

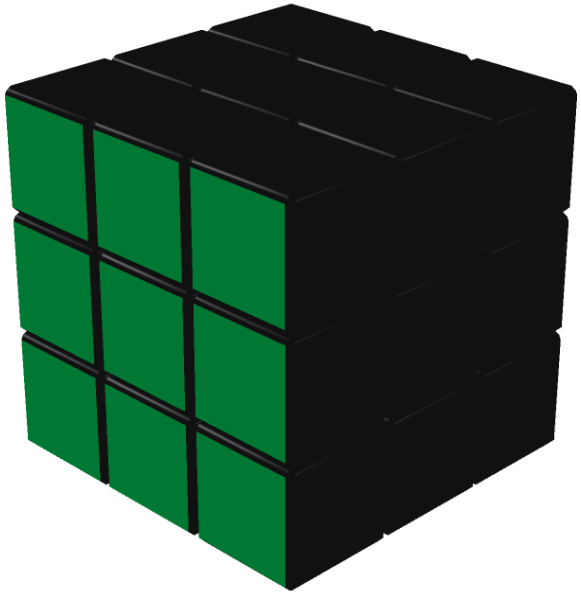


Mecanismo - Quinas

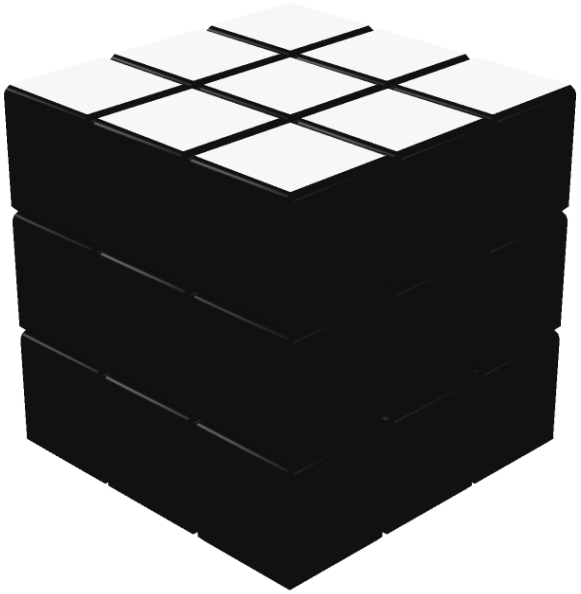
Mecanismo - Meios



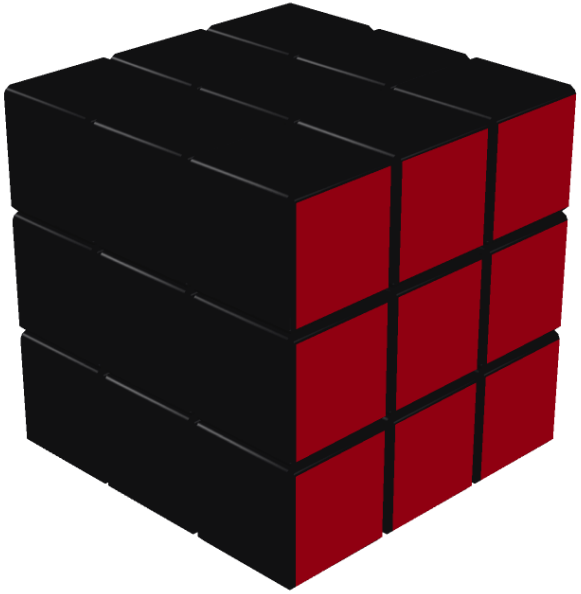
Notação - Faces



Face F (frente)
oposta à
Face B (trás)

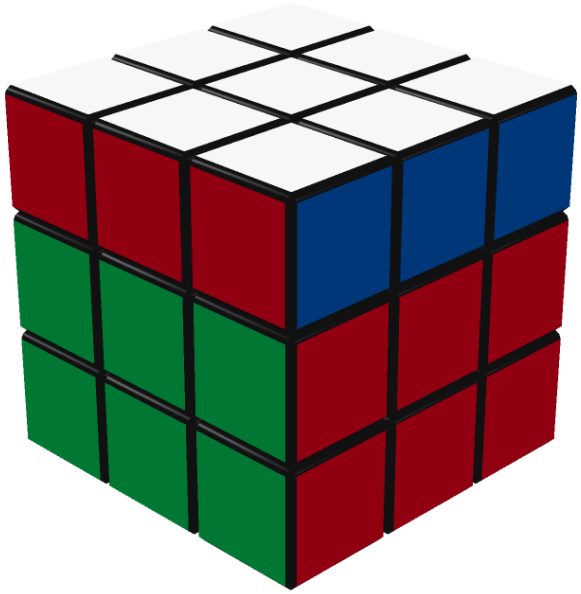


Face U (cima)
oposta à
Face D (baixo)

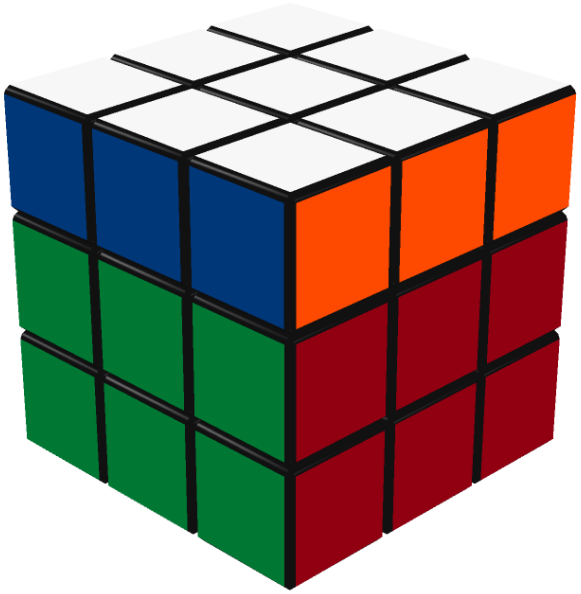


Face R (direita)
oposta à
Face L (esquerda)

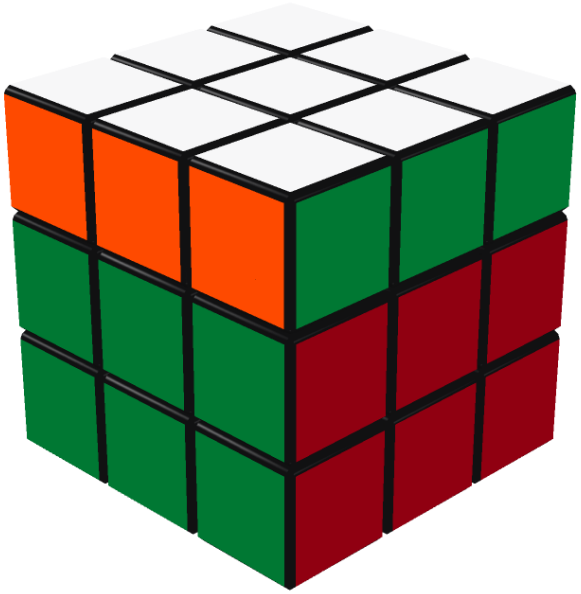
Notação - Movimentos



U (90º horário)

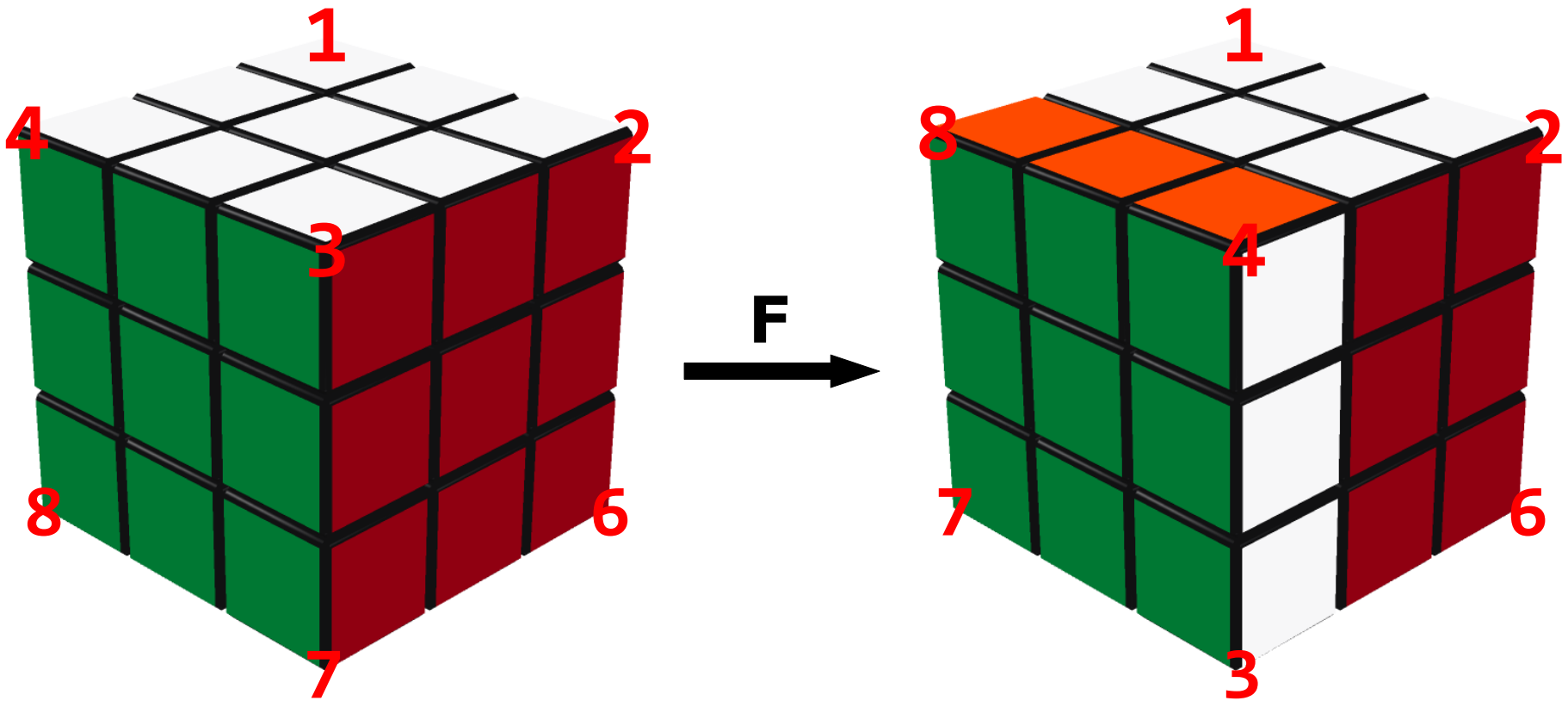


U2 (180º)



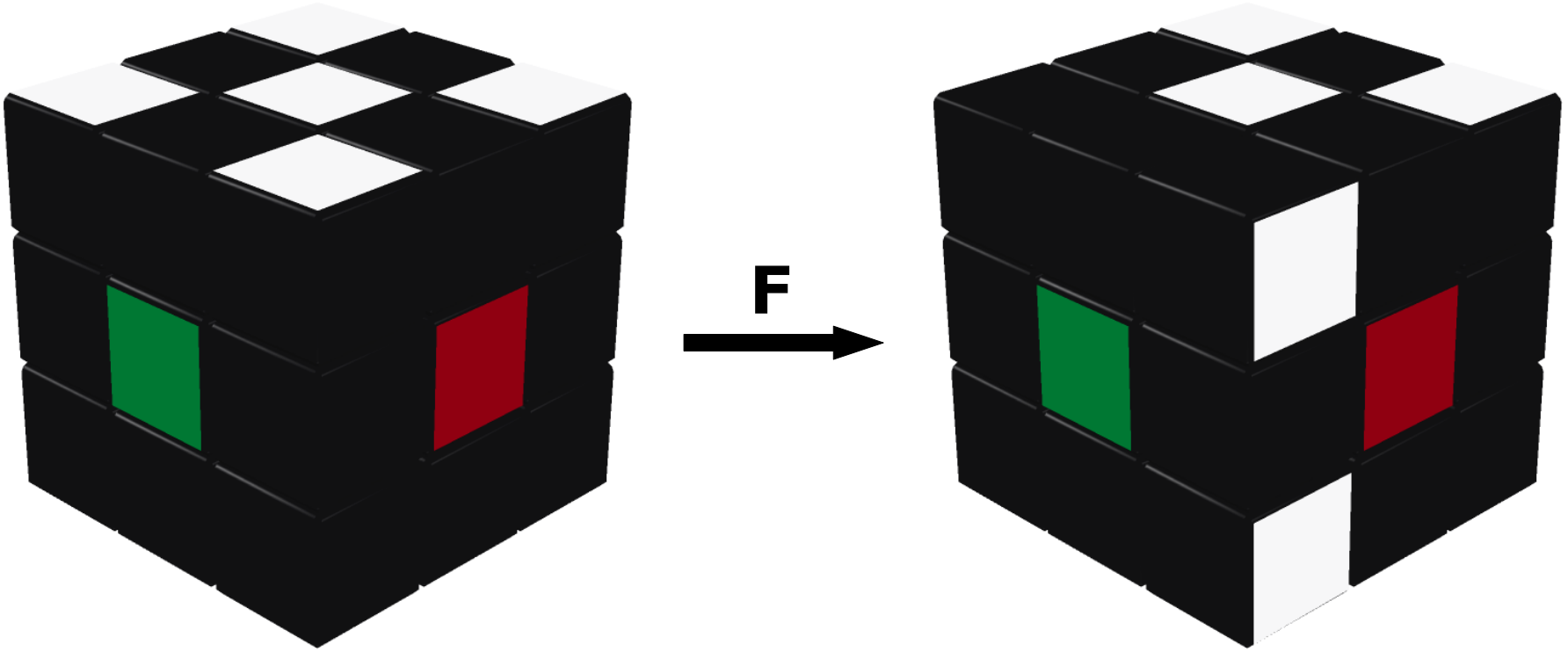
U' (90º anti-horário)

Permutação



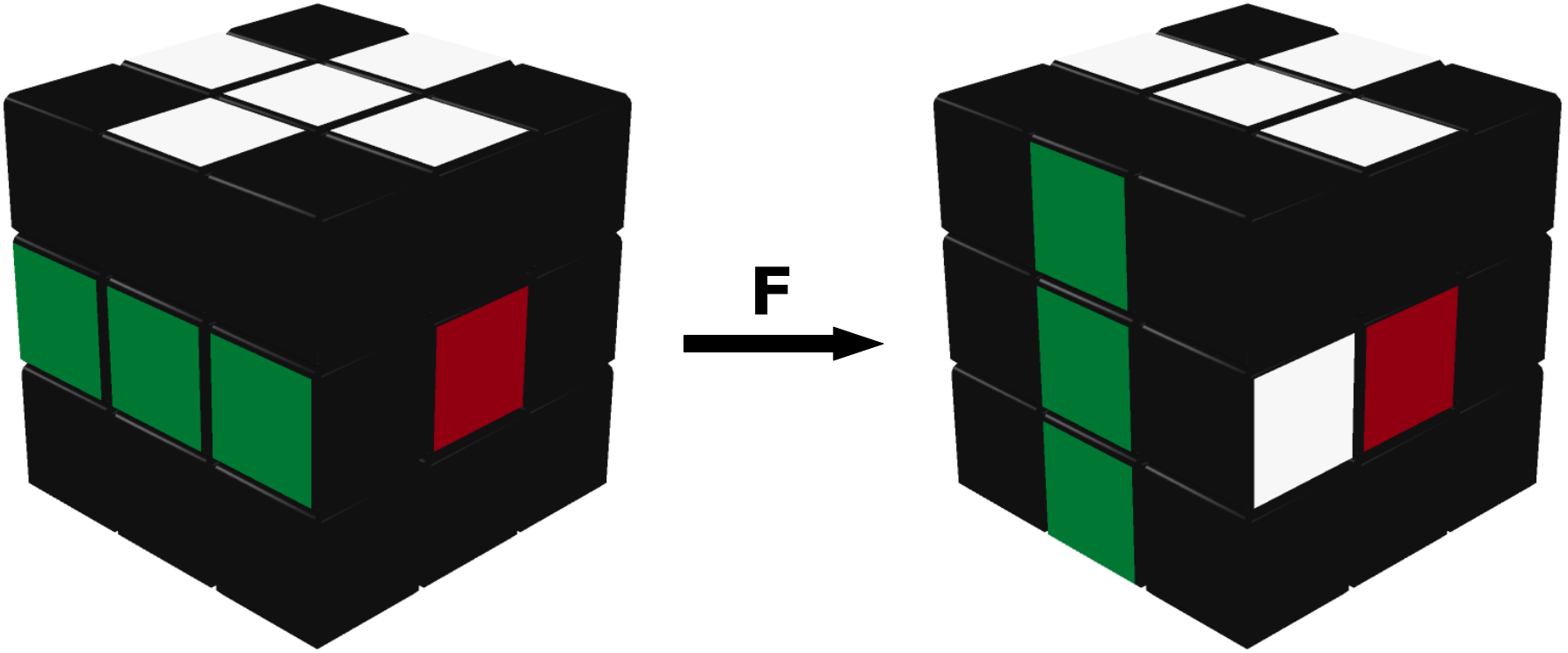
1	2	3	4	5	6	7	8
1	2	**4**	**8**	5	6	**3**	**7**

Orientação - Quinas



1	2	3	4	5	6	7	8
0	0	**1**	**2**	0	0	**2**	**1**

Orientação - Meios



1	2	3	4	5	6	7	8	9	10	11	12
0	0	**1**	0	0	0	**1**	**1**	0	0	**1**	0

Representação

```
 1  -- cubo resolvido
 2  id = {
 3    pQuinas = { 1, 2, 3, 4, 5, 6, 7, 8 },
 4    oQuinas = { 0, 0, 0, 0, 0, 0, 0, 0 },
 5    pMeios = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 },
 6    oMeios = { 0, 0, 0, 0, 0, 0, 0, 0, 0,  0,  0,  0 },
 7  }
 8
 9  -- movimento F
10  f = {
11    pQuinas = { 1, 2, 4, 8, 5, 6, 3, 7 },
12    oQuinas = { 0, 0, 1, 2, 0, 0, 2, 1 },
13    pMeios = { 1, 2, 7, 11, 5, 6, 4, 8, 9, 10, 3, 12 },
14    oMeios = { 0, 0, 1,  1, 0, 0, 1, 0, 0,  0, 1,  0 },
15  }
```

Representação - Composição

```
 1  function mul(e1, e2)
 2    local pQuinas = {}
 3    local oQuinas = {}
 4    for i = 1, 8 do
 5      pQuinas[i] = e1.pQuinas[e2.pQuinas[i]]
 6      oQuinas[i] = (e1.oQuinas[e2.pQuinas[i]] + e2.oQuinas[i]) % 3
 7    end
 8
 9    local pMeios = {}
10    local oMeios = {}
11    for i = 1, 12 do
12      pMeios[i] = e1.pMeios[e2.pMeios[i]]
13      oMeios[i] = (e1.oMeios[e2.pMeios[i]] + e2.oMeios[i]) % 2
14    end
15
16    return {
17      pQuinas = pQuinas,
18      oQuinas = oQuinas,
19      pMeios = pMeios,
20      oMeios = oMeios,
21    }
22  end
```

Tamanho do espaço de busca

Permutações de quinas	40 320
Orientações de quinas	6 561
Permutações de meios	479 001 600
Orientações de meios	4 096

Restrição de paridade	1/2
Restrição de orientação de quinas	1/3
Restrição de orientação de meios	1/2

43 252 003 274 489 856 000

Solução 1 - Busca em profundidade

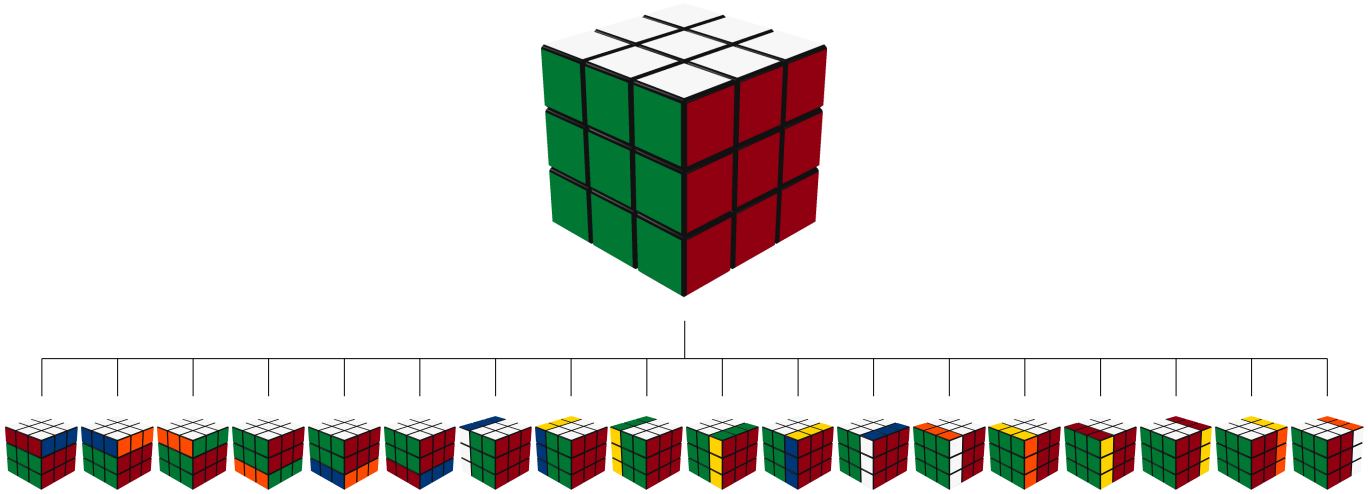
```
 1  function busca_em_profundidade(e, prof, sol)
 2    if e == id then
 3      return true
 4    end
 5
 6    if prof >= 0 then
 7      for nome, mov in pairs(movimentos) do
 8        table.insert(sol, nome)
 9        if busca_em_profundidade(mul(e, mov), prof - 1, sol) then
10          return true
11        end
12        table.remove(sol)
13      end
14    end
15
16    return false
17  end
```

Solução 2 - Busca com aprofundamento iterativo

```
 1  function busca(e, prof, sol)
 2    if prof == 0 then
 3      return e == id
 4    end
 5
 6    for nome, mov in pairs(movimentos) do
 7      table.insert(sol, nome)
 8      if busca(mul(e, mov), prof - 1, sol) then
 9        return true
10      end
11      table.remove(sol)
12    end
13  end
14
15  function busca_aprofundamento_iterativo(e)
16    for prof = 0, 20 do
17      local sol = {}
18      if busca(e, prof, sol) then
19        return sol
20      end
21    end
22  end
```

Fator de ramificação

Cada estado do cubo mágico possui 18 estados vizinhos:



Porém, algumas sequências “não fazem sentido”, por exemplo:

```
R R -> R2     R R2 -> R'     R R' -> id
```

Esta pequena observação nos permite reduzir o fator de ramificação para **15**.

Representação - Índices

Podemos agilizar algumas operações se usarmos inteiros para representar os estados.

Definindo as seguintes bijeções:

```
-- permutação <-> [0, n! - 1]
function permutacao_para_indice(p) ... end
function indice_para_permutacao(i, n) ... end


-- orientação <-> [0, b^n - 1]
function orientacao_para_indice(o, b) ... end
function indice_para_orientacao(i, b, n) ... end


-- combinação <-> [0, C(n, k) - 1]
function combinacao_para_indice(c, k) ... end
function indice_para_combinacao(i, k, n) ... end
```

Representação - Índices

Podemos converter entre as representações da seguinte forma:

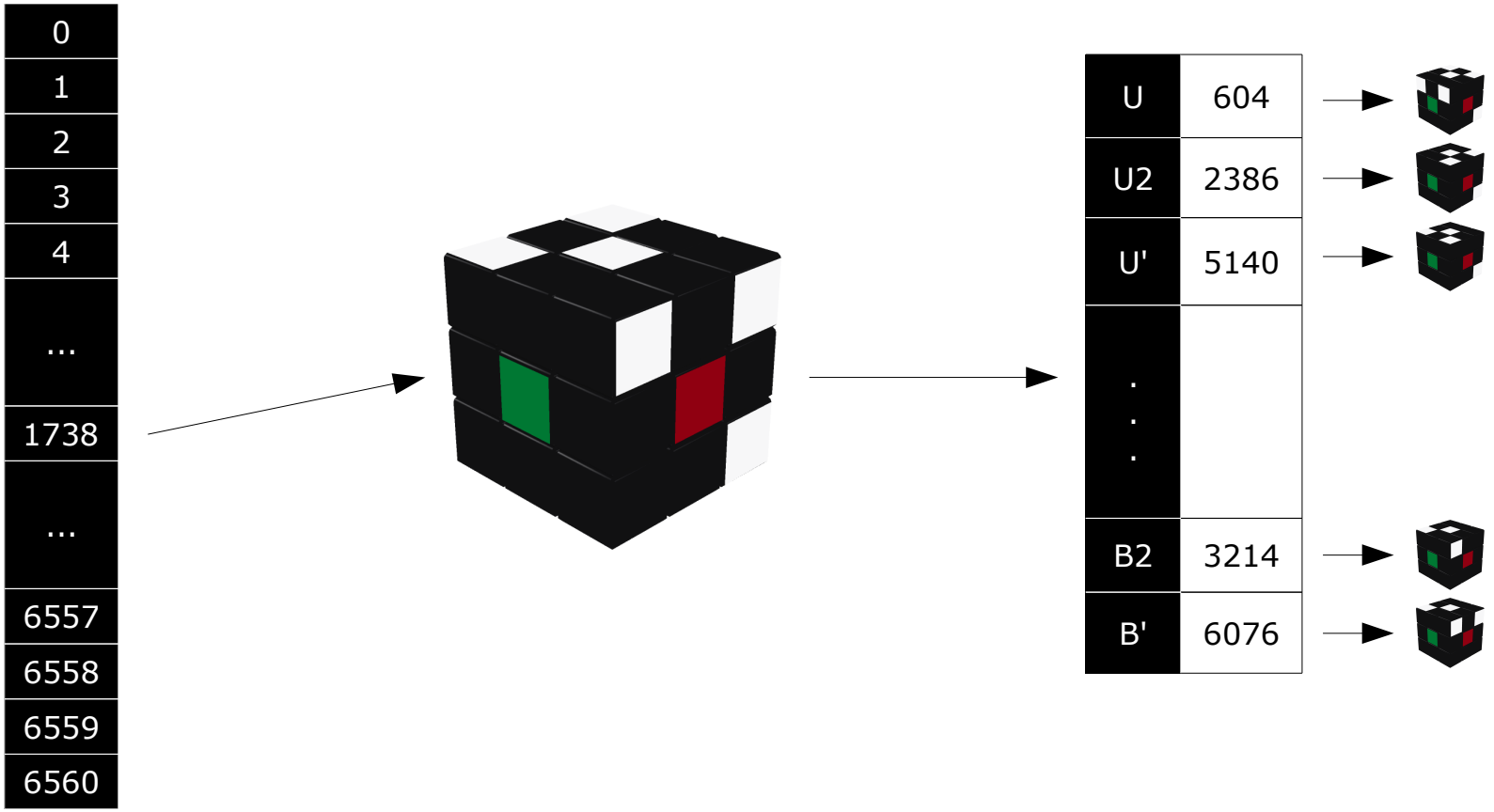
```
-- estado -> indices
function estado_para_indices(e)
  return {
    pQuinas = permutacao_para_indice(e.pQuinas),
    oQuinas = orientacao_para_indice(e.oQuinas, 3),
    pMeios = permutacao_para_indice(e.pMeios),
    oMeios = orientacao_para_indice(e.oMeios, 2),
  }
end

-- indices -> estado
function indices_para_estado(i)
  return {
    pQuinas = indice_para_permutacao(i.pQuinas, 8),
    oQuinas = indice_para_orientacao(i.oQuinas, 3, 8),
    pMeios = indice_para_permutacao(i.pMeios, 12),
    oMeios = indice_para_orientacao(i.oMeios, 2, 12),
  }
end
```

Tabelas de transição

Tabelas de transição são mapeamentos da forma `(indice, movimento) -> indice`.

Sua função é agilizar a geração das árvores de busca.



Tabelas de transição

```
 1  oQuinasTrans = {}
 2  for i = 0, 6560 do
 3    local estado = indices_para_estado({ oQuinas = i })
 4
 5    oQuinasTrans[i] = {}
 6    for nome, mov in pairs(movimentos) do
 7      oQuinasTrans[i][nome] =
 8        estado_para_indices(mul(estado, mov)).oQuinas
 9    end
10  end
```

Solução 3 - Busca com tabelas de transição

```
 1  iId = estado_para_indices(id)
 2
 3  function busca(i, prof, sol)
 4    if prof == 0 then
 5      return i == iId
 6    end
 7
 8    for nome, _ in pairs(movimentos) do
 9      local prox = {
10        pQuinas = pQuinasTrans[i.pQuinas][nome],
11        oQuinas = oQuinasTrans[i.oQuinas][nome],
12        pMeios = pMeiosTrans[i.pMeios][nome],
13        oMeios = oMeiosTrans[i.oMeios][nome],
14      }
15
16      table.insert(sol, nome)
17      if busca(prox, prof - 1, sol) then
18        return true
19      end
20      table.remove(sol)
21    end
22  end
```

Heurísticas

Os algoritmos apresentados até agora exploram toda a árvore de busca.

Para reduzir o número de nós analisados, precisamos de uma heurística, ou seja, de uma função que estime a distância entre um nó qualquer e o nó solução.

Para garantir a otimalidade das soluções, a heurística não pode superestimar a distância real.

Heurísticas - Quinas

Vamos usar como heurística o sub-problema das quinas do cubo mágico.

Considerando apenas as quinas, temos

$$8! * 3^7 = \mathbf{88\ 179\ 840}$$

estados diferentes, que podem ser explorados em poucos segundos em um PC atual.

A distribuição das distâncias é a seguinte:

0	1
1	18
2	243
3	2 874
4	28 000
5	205 416

6	1 168 516
7	5 462 528
8	20 776 176
9	45 391 616
10	15 139 616
11	64 736

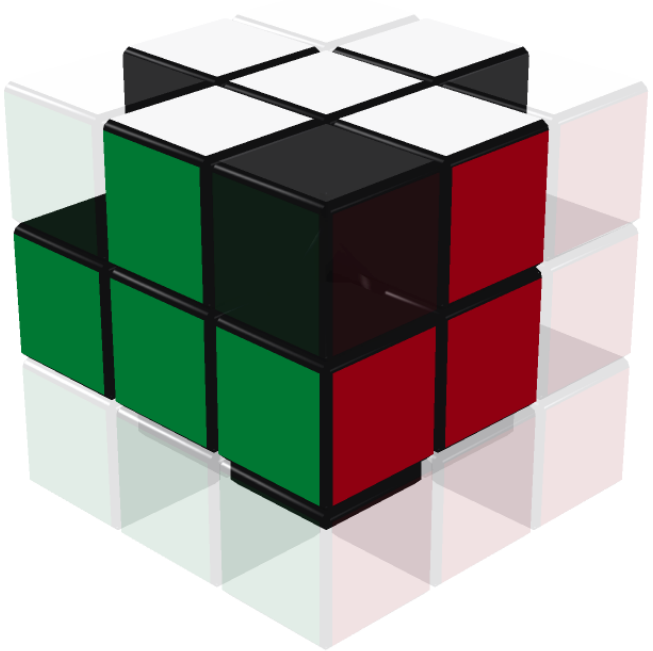
Heurísticas - Meios

Poderíamos usar como heurística o sub-problema dos meios, mas o número de estados possíveis é muito grande:

$$12! * 2^{11} = \mathbf{980\ 995\ 276\ 800}$$

O que podemos fazer é considerar dois conjuntos de seis meios. Dessa forma temos

$$2 * C(12, 6) * 6! * 2^{6} = 2 * \mathbf{42\ 577\ 920}$$



A distribuição de um conjuntos meios é a seguinte:

0	1
1	15
2	2 360
3	27 139
4	281 416

5	2 380 459
6	13 065 209
7	23 961 831
8	2 859 244
9	56

Heurísticas

```
 1  oQuinasDist = {}
 2  oQuinasDist[estado_para_indices(id).oQuinas] = 0
 3
 4  local distancia = 0
 5  local visitados = 0
 6  while visitados < 6561 do
 7    for i = 0, 6560 do
 8      if oQuinasDist[i] == distancia then
 9        for nome, _ in pairs(movimentos) do
10          local prox = oQuinasTrans[i][nome]
11          if not oQuinasDist[prox] then
12            oQuinasDist[prox] = distancia + 1
13            visitados = visitados + 1
14          end
15        end
16      end
17    end
18  end
```

Solução Final - IDA* (Korf 1997)

```
 1  function busca(i, prof, sol)
 2    if prof == 0 then
 3      return i == iId
 4    end
 5
 6    if quinasDist[i.cQuinas][i.pQuinas][i.oQuinas] > prof or
 7       meios1Dist[i.cMeios1][i.pMeios1][i.oMeios1] > prof or
 8       meios2Dist[i.cMeios2][i.pMeios2][i.oMeios2] > prof then
 9      return false
10    end
11
12    for nome, _ in pairs(movimentos) do
13      local prox = { ... }
14
15      table.insert(sol, nome)
16      if busca(prox, prof - 1, sol) then
17        return true
18      end
19      table.remove(sol)
20    end
21  end
```

Conclusões

O solucionador de Korf é capaz de resolver otimamente estados aleatórios do cubo mágico em questão de minutos num PC moderno.

As ideias apresentadas podem ser utilizadas para a resolução de outros tipos de quebra-cabeças, como o *2x2x2*, o *pyraminx*, o *skewb*, etc..

Referências

Korf, Richard. *Finding optimal solutions to Rubik's Cube using pattern databases*. 1997.

Kociemba, Herbert. *Cube Explorer*. `http://kociemba.org/cube.htm`.