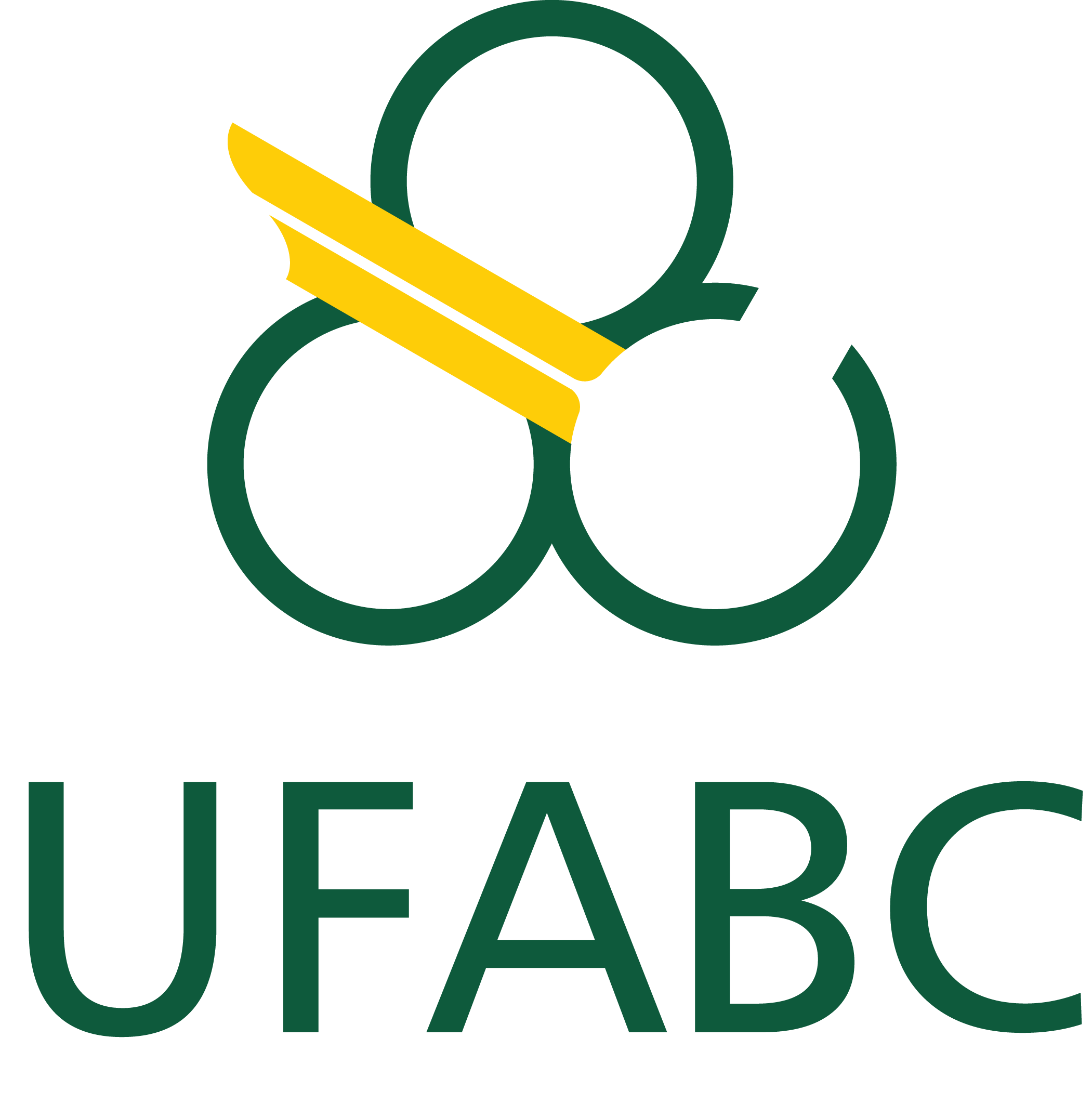
****

**ALGORITMO RESOLUÇÃO CUBO 3X3**

**Projeto POO**

**Bruno Barbosa Marques**

**RA: 11014314**

**UFABC 2015**

**Introdução**

O cubo mágico foi criado pelo professor húngaro Ernest Rubik, em 1974, e consiste num quebra-cabeça formado por 27 peças cúbicas dispostos na configuração 3 x 3 x 3. Cada eixo do cubo é rotacional, fazendo possível embaralhar as cores de cada lado. O objetivo é desembaralhar as cores, retornando o cubo ao seu estado original.

É importante mencionar que existem 43 252 003 274 489 856 000 (43 quintilhões) de combinações possíveis para o cubo mágico e, para cada caso, existe uma sequência diferente de movimentos possíveis para solucioná-lo da forma mais rápida possível. Dessa forma, utilizando os artifícios de programação até então estudados em aula, se quiséssemos criar um algoritmo que resolvesse o cubo com a menor quantidade de movimentos, teríamos de listar os 43 quintilhões de casos ou encontrar uma forma alternativa – que exigiria um tempo a mais do disponível para se pensar.

Todavia, é possível criar um programa capaz de solucioná-lo pelo método das camadas, supondo que uma das camadas já esteja montada da forma correta, ou que o usuário já saiba montá-la – afazer que não exige o conhecimento de fórmulas, sendo possível realizá-lo por pura lógica.

**Objetivo**

Mediante da impossibilidade de criar um programa que solucione o cubo com o menor número de movimentos possível, limitamos o universo de busca utilizando um método de solução avançado: Fridrich – a partir da primeira camada solucionada, o que seria útil para acostumar cubistas iniciantes a cada caso e para ajudá-los a se familiarizar com os movimentos. Portanto, o objetivo do presente projeto é criar um programa capaz de resolver o cubo pelo método Fridrich, considerando a primeira camada já resolvida.

O método da camada consiste pura e simplesmente na solução do cubo camada por camada – e é o mais indicado aos iniciantes, pois nele existem somente cinco sequências que devem ser memorizadas. O método de Fridrich segue o mesmo modelo, mas com alguns atalhos e novas fórmulas. Este método segue as seguintes etapas:

1. Escolher uma face e nela montar uma cruz com a cor respectiva do lado escolhido
2. Resolver simultaneamente a face escolhida e a camada acima dela
3. Orientar as peças da camada ainda não solucionada para cima, isto é, deixar a camada oposta à escolhida na etapa 1 com a sua cor respectiva virada para cima
4. Permutar as peças da camada não resolvida

**Notação**

Uma vez que a proposta do programa é retornar uma série de movimentos ao usuário, é fundamental determinar como serão representadas as rotações na saída.

Tomando uma das faces como a frontal, teremos as seguintes faces representadas das seguintes formas:

* Frontal (front) – F
* Trás (back)– B
* Baixo (down)– D
* Cima (up)– U
* Esquerda (left)– L
* Direita (right)– R

Nesse sentido, se o programa retornar “F” o movimento será no eixo frontal, de modo que “F” é um movimento no sentido horário e “F**’**”, no anti-horário. Além disso, há o movimento “(F->R)” que implica em girar o cubo de modo que a camada antes tida como direita, se torne a camada da frente (mantendo as camadas de cima e baixo inalteradas).

**Metodologia**

A princípio, a ideia seria representar a posição de cada peça a partir de coordenadas (x, y, z) no espaço, de modo a tornar possível a manipulação das combinações do cubo pelo computador. No entanto, mostrou-se mais viável abandonar este mecanismo, dado que para representarmos as quinas e suas possíveis orientações, teríamos de usar 24 coordenadas – e se numerássemos as quinas de 0 a 7, teríamos apenas oito valores.

Para construirmos nosso algoritmo, partimos da numeração das peças:

No momento no qual o cubo está resolvido, determinamos para as peças das quinas os números de 0 a 7, sendo 0 a peça da esquerda frontal superior, 1 sendo a peça da direita frontal superior e assim por diante (sentido horário). Além disso, os números 0, 1 e -1 determinam a orientação da peça, tal que:

* Se 0, a peça está orientada corretamente
* Se 1, a peça precisa realizar um movimento anti-horário para tornar-se correta
* Se -1, a peça precisa realizar um movimento horário para tornar-se correta

Nesse sentido, formamos a seguinte matriz para quando o cubo está resolvido:

Posição -> [0 1 2 3 4 5 6 7]

Orientação -> [0 0 0 0 0 0 0 0]

Para tanto, o programa irá operar com as peças do cubo até o momento em que a matriz construída fique da forma como foi colocada acima. O mesmo irá ocorrer com as peças do meio – e por meio, entende-se a peça localizada entre as quinas -, que irão ser numeradas de maneira análoga de 0 a 11, sendo 0 a peça superior frontal. Quanto à sua orientação, como estas peças tem apenas dois lado, possuem simplesmente orientação 0 (correto) e 1 (incorreto). Formamos a seguinte matriz:

Posição -> [0 1 2 3 4 5 6 7 8 9 10 11]

Orientação -> [0 0 0 0 0 0 0 0 0 0 0 0]

Nesse sentido, o programa começa com a inserção dos dados, ou seja, colocação das posições de cada peça seguido da orientação delas – como estamos resolvendo pelo método de Fridrich, o primeiro passo consiste em arrumar a segunda camada – portanto, consiste em trabalhar com as quinas dela, respeitando a ordem de primeiro inserir as posições e depois, as orientações.

Então, primeiramente declaramos matrizes nas quais ao inserirmos os dados, estaremos construindo a forma como o cubo está embaralhado naquele instante. E a cada movimento realizado, como o F, os valores da matriz serão alterados e, portanto, devem ser atualizados para o computador. Precisamos criar funções que façam essa atualização para cada movimento:

Este raciocínio segue análogo para cada movimento. Uma vez que temos todos os movimentos atualizados, basta utilizar operadores *while*, *if* e *else*. O *while* diz ao computador para operar até o momento em que a matriz não está formada, já os *if/else* retornam os movimentos necessários para resolver o cubo em função da forma que ele está embaralhado naquele instante.

O primeiro passo consiste em solucionar a segunda camada permutando as quinas. Para orientar a última camada e permutar suas peças, o raciocínio segue análogo nos passos 2 e 3, respectivamente, do algoritmo.

Exemplo de entrada de dados:

5 1 2 3 4 0 6 7

1 1 0 0 1 0 0 0

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

1 0 0 0 1 1 0 0 1 1 0 0