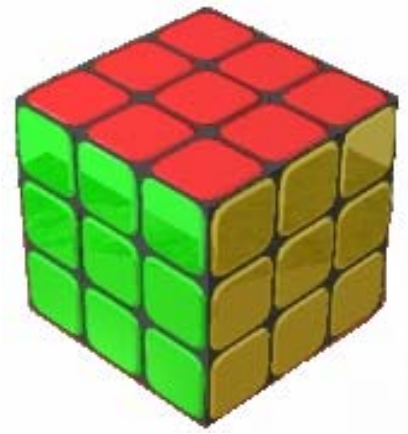
34ª Jornadas Argentinas de Informática e Investigación Operativa

"Algoritmos de resolución para el cubo de Rubik"

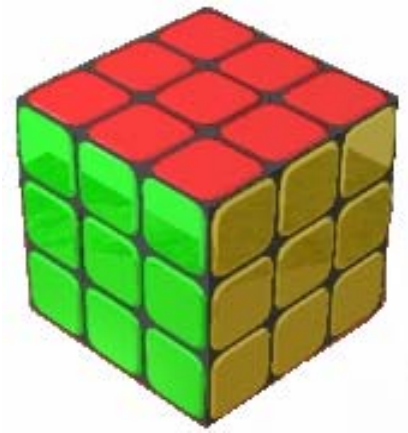


Categoría: Trabajos de Cátedra
Área: "Algoritmos, lenguajes y programación"
Autores:
Ridao Freitas, Iván Osvaldo (iridao@exa.unicen.edu.ar)
Vidal, Santiago Agustín (svidal@exa.unicen.edu.ar)
Directora del trabajo: Laura Felice

Universidad Nacional del Centro de la Provincia de Buenos Aires

34ª Jornadas Argentinas de Informática e Investigación Operativa

“Algoritmos de resolución para el cubo de Rubik”



Categoría: Trabajos de Cátedra
Área: “Algoritmos, lenguajes y programación”

Índice

RESUMEN

El objetivo de este trabajo fue diseñar e implementar un algoritmo que resuelva el problema de armar el cubo de Rubik (o cubo mágico), dada cualquiera de las más de 43 trillones de posibles configuraciones de éste. Para esto se realizó un trabajo de investigación mediante el cual se procedió a delinear los pasos para un algoritmo basado en la técnica de Backtracking. Se analizaron distintas heurísticas para reducir el espacio de búsqueda y así obtener un algoritmo más eficiente.

No estando totalmente satisfechos con esta primera solución, se implementó un segundo programa del mismo tipo, aunque mucho más complejo y sofisticado que logró reducir a la mitad la cantidad de movimientos en los cuales se arma el cubo. Se describe en este trabajo las dos soluciones propuestas y una comparación entre ambas.

Para darle una funcionalidad real al programa se le provee al usuario la posibilidad de cargar cualquier cubo mediante una interfaz gráfica amigable. Cabe destacar que esta carga va acompañada de un conjunto complejo de comprobaciones que cerciora configuraciones válidas.

Este trabajo fue realizado como proyecto final para una materia que introduce en el análisis y el diseño de algoritmos. La misma es dictada en segundo año de una carrera de Informática.

El lenguaje utilizado para la implementación fue C++ y el entorno de desarrollo Borland Builder C++.

1.INTRODUCCIÓN

El famoso cubo de Rubik fue inventado en el año 1974 por un profesor de Arquitectura de la Universidad de Budapest, en Hungría, llamado Erno Rubik [10]. El propósito fue explicar a sus alumnos algunos conceptos relacionados con el volumen y el espacio, pero luego de algún tiempo el juego se hizo tan famoso que fue lanzado al mercado internacional.

El cubo de Rubik es un bloque cúbico con su superficie subdividida de modo que cada cara consiste en nueve cuadrados. Cada cara se puede rotar, dando el aspecto de una rebanada entera del bloque que rota sobre sí mismo. Esto da la impresión de que el cubo está compuesto de 27 cubos más pequeños ($3 \times 3 \times 3$). En su estado original cada cara del cubo es de un color, pero la rotación de cada una de estas permite que los cubos más pequeños sean combinados de muchas maneras. Tal es así que el cubo puede tener más de 43 trillones de diversas posiciones. Esto deriva de que, por una parte podemos combinar entre sí de cualquier forma todas las esquinas, lo que da lugar a 8! posibilidades. Con las aristas pasa lo mismo, es decir, que podemos combinarlas de todas las formas posibles lo que da lugar a 12! posibilidades, pero la permutación total de esquinas y aristas debe ser par lo que nos elimina la mitad de las posibilidades. Por otra parte, podemos rotar todas las esquinas como queramos salvo una sin cambiar nada más en el cubo. La orientación de la última esquina vendrá determinada por la que tengan las otras siete y esto nos crea 3^7 posibilidades. Con las aristas pasa lo mismo, es decir, nos aparecen 2^{11} posibilidades más. En total tendremos que el número de permutaciones posibles en el Cubo de Rubik es de:

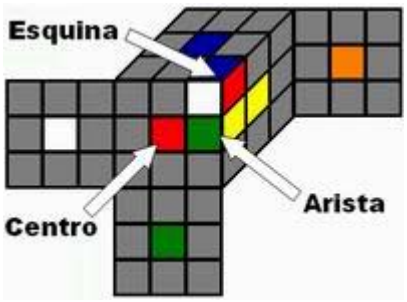
$$\mathbf{(8!\ 12!\ 3^7\ 2^{11})/2 = 43.252.003.274.489.856.000} \qquad [5]$$

1.1 *Objetivo del juego*

El objetivo básico del juego es restaurar el cubo a su condición original. Se deben utilizar las diferentes rotaciones que el cubo permite, en cada uno de sus lados, para ir llevando cada pieza de éste a su correcta ubicación, logrando así que cada cara sea de un único color.

1.2 *Definiciones básicas*

1.2.1 Las piezas del cubo

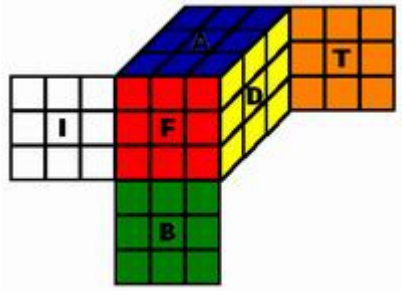


El cubo está formado por 3 clases distintas de piezas: las centrales, las aristas y las esquinas. Cada una de estas piezas se caracteriza porque poseen 1, 2 ó 3 colores respectivamente. Es importante notar que en realidad son las aristas y las esquinas las que se mueven, pues las piezas centrales siempre guardan la misma posición relativa entre ellas. Todos los movimientos que pueden hacerse con el cubo se reducen a girar una o más veces las caras del cubo, sin desplazar de su posición las piezas centrales.

1.2.2 Notación y movimientos básicos

Lo primero por realizar a la hora de expresar soluciones para el cubo es efectuar una notación adecuada; por lo que se nombra cada cara con una letra mayúscula para describir los movimientos sobre el cubo:

La cara de Arriba: A
La cara de aBajo: B
La cara de la Izquierda: I
La cara de la Derecha: D
La cara del Frente: F
La cara de aTrás: T



Cuando se nombre una cara por su letra, va a significar (en términos de movimiento) un giro de un cuarto de vuelta (90 grados) en la dirección de las agujas del reloj. Es decir si decimos el movimiento A, esto quiere decir un giro de un cuarto de vuelta de la cara de Arriba, en el sentido de las agujas del reloj. Del mismo modo, el movimiento contrario (es decir, un giro de un cuarto de vuelta en el sentido contrario a las agujas del reloj) estará indicado por A'. En tanto que, un giro de media vuelta (180 grados) se indica como A2. Es importante aclarar que hacer A, A’ o A2 equivale, en nuestra notación, a realizar un único movimiento.

Una secuencia de movimientos es una serie de letras mayúsculas que representan giros que deben aplicarse sucesivamente sobre el cubo. Así, la secuencia DAD' significa un giro de la cara Derecha de 90 grados en el sentido de las agujas del reloj, seguido de un giro de la cara de Arriba de 90 grados en el mismo sentido, seguido de un giro de 90 grados de la cara Derecha en sentido contrario a las agujas del reloj.

Este artículo está organizado de la siguiente manera. La sección 2 introduce brevemente la metodología utilizada para el desarrollo del programa. La sección 3 explica detalles de la implementación con las clases utilizadas y se describen los algoritmos implementados para la resolución del cubo. En la sección siguiente se detallan otros aspectos importantes en cuanto a la funcionalidad del programa. Por último, en la sección 5 se presentan las conclusiones.

2.METODOLOGÍA

La metodología aplicada se basa en una disciplina de diseño a través de los tipos de datos abstractos y una disciplina de desarrollo con técnicas de programación rigurosas.

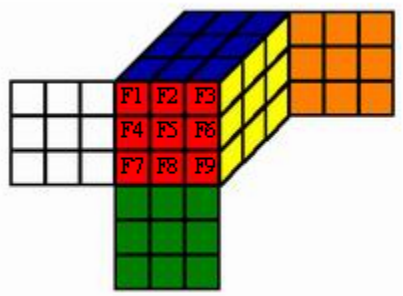
La construcción de algoritmos eficientes requiere para comenzar que los niveles de especificación que describen un problema sean independientes de la representación de los tipos de datos y del lenguaje específico. La especificación debe permitir que el programador haga un

análisis predictivo de la complejidad temporal y espacial, ayudándole a elegir una implementación para las clases de objetos que participan.

De esta manera es posible introducir nociones básicas de las estructuras de datos y de los algoritmos (complejidad del algoritmo, técnicas de diseño del algoritmo, y los algoritmos clásicos extensos) en un marco que permita incluir elementos orientados a objetos, especificaciones formales y reutilización de componentes. Este marco es no solamente adecuado para programas a pequeña escala, sino que también, permite introducir principios del diseño del software de validez en la programación a gran escala.

3.IMPLEMENTACIÓN

La representación utilizada para el cubo consta de 6 matrices de 3x3 que contienen la información de cada cara. Cada cuadro está rotulado con una letra y un número que lo identifica unívocamente. De este modo será, como puede verse en el ejemplo, que cada cuadrado de la cara "frente" tiene una letra F y un "índice" entre 1 y 9.



Cuando se realiza una rotación de una cara (cualquiera sea ésta), se ve claramente que se produce una modificación no solamente en la matriz que la representa, sino también, en las matrices de sus adyacentes.

3.1 *Diseño de clases*

Para la resolución del problema se decidió modularizar el mismo y diseñarlo a través de la utilización de clases, las cuales se describen brevemente a continuación:

- **Clase Principal:** Esta clase se encarga de la interfaz gráfica y de manejar todo aquello que el usuario puede pedirle al programa.

- **Clase Gráficos:** Esta clase se encarga de dibujar en pantalla el cubo según la información que se encuentra en las matrices y de las distintas comprobaciones cuando este se carga manualmente.

- **Clase Cubo:** Esta clase se encarga de contener la información del cubo y los métodos relacionados con éste, es decir, aquí se definen las seis matrices, las diferentes rotaciones y los métodos para resolver el cubo.

- **Clase AlgoritmoPrincipiante y clase AlgoritmoExperto:** Estas clases encapsulan los métodos que se realizan para cada uno de los algoritmos.

3.2 *Desarrollo de la solución*

La aplicación cuenta con dos algoritmos que se usan alternativamente según la intención del usuario. En un primer momento se desarrolló el algoritmo al cual hemos llamado **Principiante**, pero viendo que se alcanzaban soluciones en una cantidad de pasos excesiva, se desarrolló un segundo algoritmo, al que llamamos **Experto**, el cual en la mayoría de los casos, encuentra soluciones en la mitad de pasos que su antecesor.

Ambos algoritmos se han construido como algoritmos por pasos, es decir, se les pide cumplir con ciertos requisitos, según el caso, para avanzar en la "línea de resolución". No obstante vale aclarar que el algoritmo de Backtracking es único, no estando éste sectorizado y que,

Principiante y Experto son las diferentes heurísticas que va a utilizar el Backtracking para encontrar la solución. Estas heurísticas son complejas y se traducen en reconocer estados puntuales del cubo, ante los cuales, seguir pequeñas listas de movimientos llevan a una rápida localización del paso siguiente. Por supuesto esta técnica conlleva un aumento muy importante en la velocidad de respuesta del programa, que es de apenas segundos.

Es importante comentar que para abordar el problema, lo primero en construirse fue un Backtracking en el que la prioridad fue que no se pasara por estados repetidos del cubo con lo cual se cortó una innumerable cantidad de ramas, pero como es de suponer, esto no alcanzó para llegar a soluciones del problema. Luego se trabajó con un algoritmo de profundidad 20. Se adoptó esto por nuestro conocimiento de una teoría (aún no probada matemáticamente) que afirma que cualquier cubo puede ser resuelto en no más de 20 movimientos [13]. Puede suceder pensar equivocadamente que construyendo un Backtracking de profundidad 20 bastaría para llegar a soluciones. Esto es cierto, pero también de la teoría deriva de forma directa que un Backtracking de 20 niveles tiene una cantidad de hijos que supera los 43 trillones. Viéndonos ante tal situación se trató de reconocer estados del cubo para los cuales puntualmente con dos o tres movimientos estos se resolvían. Esto no alcanzó por lo cual optamos por la solución heurística que no es otra cosa mas que un algoritmo que produce con rapidez soluciones buenas, pero no necesariamente óptimas.

Tanto el **Algoritmo Experto** como el **Principiante** surgen de una investigación vía Internet en la cual se encontraron diversas estrategias de resolución. Después de interiorizarnos profundamente con cada una de las soluciones propuestas y de mezclar y probar entre varios métodos se arribó a los algoritmos propuestos.

A continuación se presenta un pseudocódigo con estructura C del esquema general del backtracking:

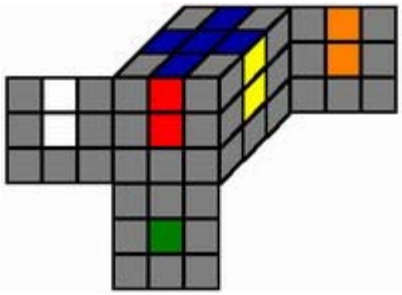
```
void backtracking()
{
  if((mejor_solucion.lenght()==0) || (paso==1))
  {
    if(paso==1)
      reconocer_si_faltan_dos_movimientos();
    if((hay_solucion()) || (termino_Algun_Paso()))
    {
      if((mejor_solucion.lenght()==0)
      || (mejor_solucion.lenght() > solucion.lenght()))
        actualizar_mejor_solucion();
    }
    else
    {
       int nro_hijo=1;
       Movimiento mov;
       generar_hijos_con_poda();
       obtener_hijo(nro_hijo,&mov);
       while(mov!=null)
       {
          mover_cara(mov);
          solucion.add(mov);
          backtracking();
          solucion.delete(solucion.lenght());
          deshacer_mover_cara(mov);
          nro_hijo++;
          obtener_hijo(nro_hijo,&mov);
       }
    }
  }
}
```

La función `reconocer_si_faltan_dos_movimientos()` comprueba si se cumplen los patrones necesarios para solucionar el cubo con dos movimientos. Con esta heurística se ahorra, en un número importante de situaciones, realizar el árbol de búsqueda.

La función `termino_Algun_Paso()` reconoce si se han cumplido las condiciones necesarias para dar por terminado el paso en el que se encuentra el algoritmo.

La función `generar_hijos_con_poda()` es la encargada de obtener el conjunto mínimo de los posibles movimientos que se podrán llevar a cabo, de acuerdo a la configuración del cubo en esa instancia. Es aquí donde intervienen de forma absoluta las heurísticas que se detallan en los puntos siguientes.

3.3 *Paso 1: Obtención de la Cruz*



Este primer paso es común para ambos algoritmos y consiste, en crear una cruz en una cara. Para esto basta llevar las cuatro aristas, de dicha cara, a su posición. Obsérvese que aparte de formar la cruz se debe de tener en cuenta que las aristas tienen dos colores, un color es el de la cara superior y el otro debe coincidir con el color de la cara en común. Vale aclarar que al implementar la solución decidimos elegir la cara en la que, con menor cantidad de pasos, se llegará a cumplir la etapa. Esto se logra haciendo complejas comprobaciones para cada cara que efectúan un sistema de pago. El mismo verifica la ubicación de las aristas correspondientes a cada cara y su respectiva orientación; cuando esta verificación tiene un resultado positivo, con el fin de agilizar el backtracking se procede a restringir movimientos contraproducentes para el armado de la cruz.

Por buscar la cruz de menor costo es que este paso del proceso es el que más se asemeja a un Backtracking “clásico” (sin heurística pero con poda) y es por esta razón el tiempo de demora que se produce al pedir una solución. Luego se deriva la importante afirmación de que el costo computacional del algoritmo de resolución del cubo es igual al costo computacional de este paso.

Cantidad de movimientos esperados: 7

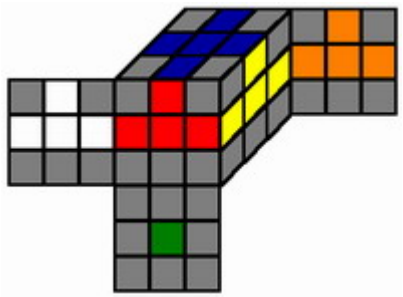
3.4 *Análisis de complejidad temporal*

Como se dijo anteriormente el paso que mayor tiempo de proceso insume es el armado de la cruz. Esta puede armarse en 7 movimientos, pero es bueno repetir que en este paso se intenta buscar la cruz a la que se llegue con menor cantidad de movimientos. Es decir, el Backtracking busca soluciones hasta localizar una aceptable según nuestro criterio. El código utilizado dentro del algoritmo no son mas que comprobaciones y heurísticas (en el caso que se cuente con piezas de la cruz bien ubicadas) por lo que la complejidad temporal depende del lapso que se tarde en encontrar una cruz adecuada (es decir, que se adapte a nuestro numero máximo de movimientos para armar ésta). Según nuestros cálculos la complejidad temporal es de $\mathbf{O(j^k)}$ donde j es la cantidad de movimientos (hijos) posibles y k la cantidad de niveles del Backtracking (que más concretamente es, en nuestro caso, la cantidad final de movimientos que se necesitan para dar por concluido el paso). Vale aclarar que este j cambia de un nivel a otro con una variación que tiende siempre a disminuir a medida que aumenta el k, esto es, debido a las podas y restricciones que se generan al ir ubicando las aristas en la cruz.

3.5 *Algoritmo Principiante*

Este algoritmo es simple y se siguen 7 pasos básicos para resolver el cubo. La simpleza radica en que se ven claramente los pasos a seguir del algoritmo y algunos movimientos podrían parecer intuitivos, aunque no su resolución. El método original en el que nos basamos es atribuido a Czes Kosniowski. Una descripción detallada del mismo se puede encontrar en [6].

3.5.1 Paso 2: Las Aristas Centrales

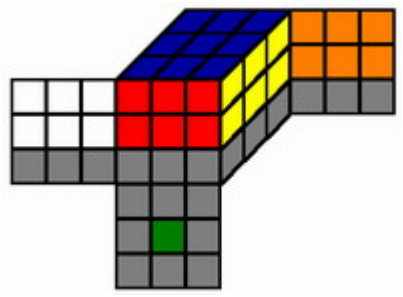


Como su nombre lo indica esta etapa consiste en colocar las aristas en la capa central del cubo. Esto se logra llevando una a una las aristas hasta su posición.
En el momento de implementar este paso se opto por seguir una heurística que marca que deben buscarse las aristas en la cara opuesta a la de la cruz y subir estas a su ubicación correspondiente por medio de tres movimientos que son similares para cada arista. En el peor de los casos, cuando no encontramos aristas para reubicar se debe hacer una rotación de la cara opuesta a la cruz o bajar una arista mal ubica para su posterior reubicación.
Cantidad de movimientos esperados: 4x3

Pseudocódigo del esquema general del Paso 2:

```
Ubica una arista en una llamada de backtracking
   Si (falta ubicar correctamente alguna arista)
   {
      Si (hay una arista en la cara opuesta a la cruz)
         Buscar secuencia de movimientos para subir arista
      Sino
         Buscar secuencia de movimientos para reubicar arista
      Aplicar secuencia de movimientos
   }
```

3.5.2 Paso 3: La Cara Superior



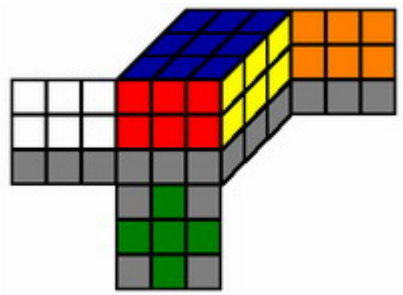
Esta etapa consiste en terminar la cara superior. Para esto basta con colocar las cuatro esquinas de esta cara en su sitio; claro que sin perder lo que se encuentra armado hasta el momento, lo cual lo hace más complicado.
La implementación de este paso se hizo en forma similar al del paso anterior. Primero se identifican las cuatro esquinas a ser ubicadas y luego, según el lugar donde se encuentran, se procede a subir estas por medio de 6 movimientos que varían para cada esquina. Nuevamente nos encontramos con casos en los que habrá que rotar la cara inferior o bajar esquinas para su posterior reubicación.
Cantidad de movimientos esperados: 4x6

Pseudocódigo del esquema general del Paso 3:

```
Ubica una esquina en una llamada de backtracking
   Si (falta ubicar correctamente alguna esquina)
   {
      Si (hay una esquina en la cara opuesta a la cruz)
         Buscar secuencia de movimientos para subir esquina
      Sino
```

```
         Buscar secuencia de movimientos para reubicar esquina
      Aplicar secuencia de movimientos
   }
```

3.5.3 Paso 4: La Cruz en la cara Inferior

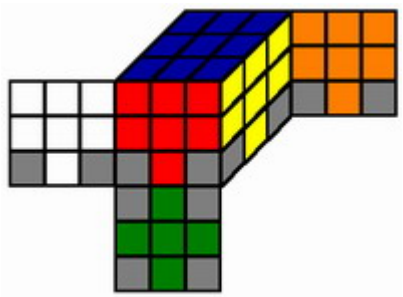


En esta etapa se pasa a la cara inferior. El objetivo es que en esta cara quede dibujada una cruz. No se debe confundir con el paso 1, ya que no se quiere que cada arista esté colocada en su sitio, sino que sólo se busca que en la cara inferior se vea la cruz. En este caso, no vamos a hacerlo poniendo una arista primero y después otra (de hecho es imposible hacerlos así) sino que lo que vamos a hacer es ponerlas de 2 en 2. Los movimientos a realizar dependerán de la posición de las aristas que ya se encuentren en la cara inferior.
Al implementar nos encontramos con que podía ocurrir que llegado este paso hubiese 4, 2 o 0 piezas bien colocadas. Para el caso de que no halla ninguna pieza en su lugar se aplican movimientos para subir 2 piezas por vez. Cuando se tienen 2 piezas en la cara inferior estas pueden ser adyacentes o inversas por lo que las secuencias de movimientos posibles también serán 2.
Cantidad de movimientos esperados: 8

Pseudocódigo del esquema general del Paso 4:

```
Orienta dos aristas en una llamada de backtracking
   Identificar cantidad de aristas en la cara inferior
   Si (la cantidad es menor a 4)
   {
      Buscar secuencia de movimientos para bajar dos aristas
      Aplicar secuencia de movimientos
   }
```

3.5.4 Paso 5: Colocación de las aristas en la cara Inferior



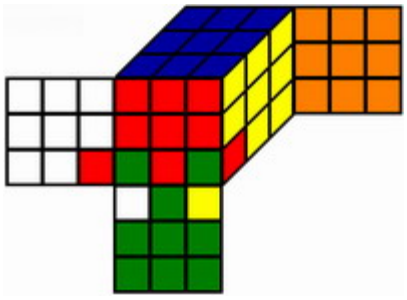
El objetivo ahora es conseguir que la cruz esté bien colocada, es decir, que las aristas se coloquen en su sitio. Nuevamente los movimientos a realizar dependerán de la posición de las aristas que ya se encuentren correctamente ubicadas.
En el momento de implementar deducimos rápidamente que hay dos posibles casos: que halla dos aristas bien ubicadas o que las cuatro estén en la posición correcta, con lo cual se saltea el paso. Cuando hay dos aristas bien ubicadas puede ocurrir nuevamente que estas sean adyacentes u opuestas, para cada caso se aplican secuencias de movimientos distintas.
Cantidad de movimientos esperados: 12

Pseudocódigo del esquema general del Paso 5:

```
Ubica correctamente dos aristas en una llamada de backtracking
   Identificar cantidad de aristas bien ubicadas en la cara inferior
   Si (la cantidad no es 4)
   {
      Buscar secuencia de movimientos para ubicar dos aristas
```

```
      Aplicar secuencia de movimientos
   }
```

3.5.5 Paso 6: Colocación de las esquinas en la cara Inferior

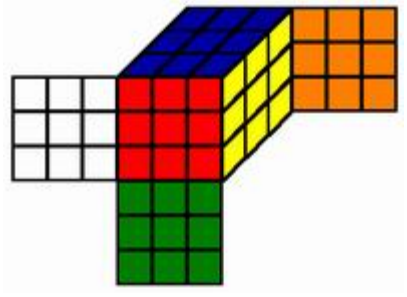


Este paso consiste en colocar las esquinas de la cara inferior en su sitio aunque posiblemente queden giradas (como se ve en el grafico). Puede observarse en el dibujo que cada esquina está en su sitio aunque dos de ellas necesitan un giro para que estén correctamente situadas.
Para implementar debimos hacer hincapié en cuántas piezas están colocadas en el lugar correcto, puede pasar que ninguna esquina esté bien ubicada; que sólo sea una o que estén ubicadas bien las cuatro con lo que se saltea el paso. Una vez identificado esto procedimos a aplicar las heurísticas según el caso, por ejemplo, cuando no se encuentre ubicada ninguna esquina las posibles correcciones pasan por rotar horizontal o diagonalmente las esquinas con respecto a la cara inferior. Cuando se encuentre que solo una esquina esta posicionada correctamente, hallamos que solo basta rotar las esquinas restantes en sentido horario o antihorario según convenga.
Cantidad de movimientos esperados: 11

Pseudocódigo del esquema general del Paso 6:

```
Posiciona correctamente las 4 esquinas en una llamada de backtracking
   Identificar cantidad de esquinas bien posicionadas en cara inferior
   Si (la cantidad no es 4)
   {
      Buscar secuencia de movimientos para posicionar las esquinas
      Aplicar secuencia de movimientos
   }
```

3.5.6 Paso 7: Orientación de las esquinas en la cara Inferior



En este paso se orientan las esquinas. Puede ocurrir que estén bien orientadas cero, una o dos esquinas y en cada caso se ubican de dos en dos una o dos veces. También puede ocurrir que el cubo ya halla quedado armado.
Al implementar encontramos que los movimientos necesarios para realizar este paso son los de mayor longitud ya que es difícil manipular el cubo sin perder lo que hemos armado hasta este momento. Una vez que identificamos en cual de las posibles situaciones nos encontramos se aplica la secuencia de movimientos correspondiente.
Cantidad de movimientos esperados: 24

Pseudocódigo del esquema general del Paso 7:

```
Orienta correctamente 2 esquinas en una llamada de backtracking
   Identificar cantidad de esquinas bien orientadas en cara inferior
   Si (la cantidad no es 4)
   {
      Buscar secuencia de movimientos para orientar las esquinas
      Aplicar secuencia de movimientos
   }
```

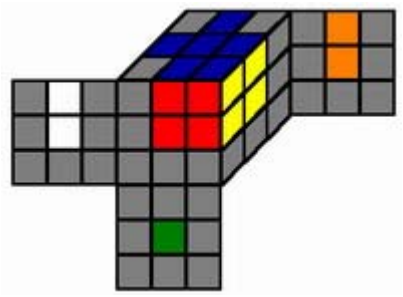
3.5.7 Promedio General

A partir de una muestra aleatoria de 500 configuraciones diferentes del cubo pudimos encontrar que la esperanza matemática (o promedio) para este algoritmo es de **101 pasos** aproximadamente.

3.6 *Algoritmo Experto*

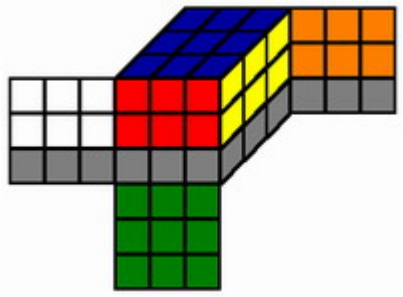
El algoritmo que se utiliza aquí es uno de los más rápidos para resolver el cubo de Rubik. Este método es debido entre otros, a Jessica Fridrich [4] y el mismo consiste en resolver el cubo en 4 pasos. Resulta entonces muy complejo, ya que utiliza una combinación de varios algoritmos que permiten minimizar la solución. A continuación se describen los pasos.

3.6.1 Paso 2: Orientación de la capa Superior y Media



Esta etapa es equivalente al paso 2 y al paso 3 del algoritmo principiante, con la diferencia que los pasos se realizan simultáneamente, por lo tanto se ahorra gran cantidad de movimientos.
Al implementar se nos presentaron 22 casos diferentes, sin contar reflexiones[1], según las posiciones de aristas y esquinas a acomodar. Por lo que el primer objetivo va a ser identificar en cual de todos los casos nos encontramos mirando la orientación y posición de una arista, en un principio, y luego de su respectiva esquina. Una vez cumplido esto se aplica la secuencia de pasos correspondiente. Como se ve en el dibujo, las aristas centrales y las esquinas se van acomodando de a pares por lo que resulta evidente que este proceso se realiza 4 veces.
Cantidad de movimientos esperados: 4x7

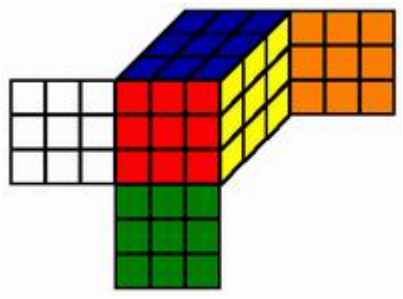
3.6.2 Paso 3: Orientación de la capa Inferior



Se comienza en esta etapa con la cara inferior. Se debe lograr que esta cara quede de un único color por lo que se busca orientar cada pieza.
La implementación de este paso reconoce en un primer momento cuantas piezas se encuentran bien ubicadas en la cara inferior, como así también el lugar que estas ocupan. Del mismo modo, se procede con las piezas restantes que se encuentran en las caras laterales. Sin contar reflexiones, pueden aparecer 40 casos distintos. Una vez identificado el caso se procede a aplicar la sucesión de movimientos correspondiente.
Cantidad de movimientos esperados: 9

[1] Reflexión: entendemos por reflexión aquellos casos isomórficos en los que se produce el efecto “espejo” en cuanto a las posiciones en X e Y de las esquinas y aristas en una cara. Además debe cumplirse que estos casos no sean simétricos ya que estaríamos ante cubos iguales.

3.6.3 Paso 4: Permutación de la capa Inferior



En esta etapa sólo hace falta permutar las piezas de la última capa sin girarlas y se obtiene el cubo terminado.
En el momento de implementar este paso tuvimos que identificar cuales deberían ser los intercambios entre aristas y esquinas para terminar de armar el cubo. Sin contar reflexiones, tras girar la cara superior se pueden presentar 13 casos distintos. Una vez reconocido el caso se sigue la secuencia de movimientos correspondiente.
Cantidad de movimientos esperados: 12

3.6.4 Promedio General

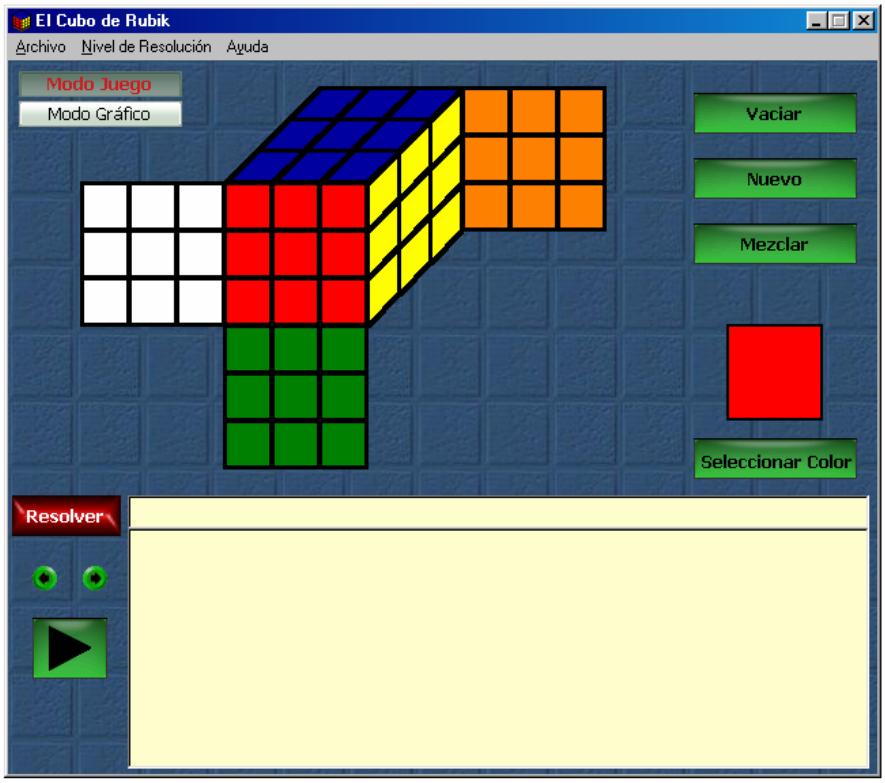
A partir de una muestra aleatoria de 500 configuraciones diferentes del cubo pudimos encontrar que la esperanza matemática (o promedio) para este algoritmo es de **56 pasos** aproximadamente.

4. OTROS ASPECTOS IMPORTANTES

Para que el programa fuera funcional en el mundo real buscamos brindar distintas posibilidades, por ejemplo cargar una configuración de cubo a elección del usuario; esto acompañado de una interfaz gráfica amigable.

Para no cargar configuraciones erróneas del cubo se cuenta con complejas funciones de validación que se ejecutan en el momento que el usuario esta cargando el cubo, y otras en el momento que el cubo esta siendo resuelto.

En la figura siguiente se muestra la pantalla principal de la aplicación. En ella puede apreciarse el cubo y las distintas opciones que brinda la herramienta.



Interfaz gráfica del programa

En la sección “Modo Juego” se accede a la posibilidad de:

- Rotar libremente las caras del cubo para obtener distintas configuraciones de éste.
- Obtener una lista de movimientos que llevan a la solución del cubo (según el nivel de resolución seleccionado). La lista a seguir se mostrará en el recuadro inferior y contendrá, además, información relevante como la cantidad de movimientos. Una vez obtenida, se podrá seguir “paso a paso” hasta completar el cubo.

En la sección “Modo Gráfico” se accede a la posibilidad de:

- "Vaciar" el cubo para colorearlo con una nueva configuración.
- Restaurar el cubo a su estado original.
- Obtener alguna de las posibles configuraciones del cubo, de manera aleatoria.
- Cambiar el color de alguna de las caras del cubo.
- Pintar el cubo.

5. CONCLUSIONES

Podemos recalcar como aspectos interesantes del trabajo la investigación que llevamos a cabo en un primer momento para recabar información de un tema para nosotros con muchos puntos oscuros en ese entonces. Uno de los aspectos que más nos motivó a profundizar en las soluciones es la inexistencia en Internet de programas que resuelvan eficientemente el cubo brindando los servicios y métodos que nuestro programa ofrece y con especial énfasis en la aplicación de la técnica de Backtracking.

Se prevé realizar una demostración del programa mediante la carga y resolución de diversas configuraciones del cubo, de especial interés para el análisis del funcionamiento de los algoritmos.

6. REFERENCIAS

[1] Aho, A. Ullman, J. “ Foundations of Computer Science” C Edition. Computer Science Press. 1995.

[2] Cormen, T.; Lieserson, C.; Rivest, R. “Introduction to Algorithms” Ed. The MIT Press. 1990.

[3] Baase, S. “Computer Algorithms. Introduction to Design and Analysis” Second Edition. 1993.

[4] Jessica Fridrich. Rubik’s Cube. http://www.ws.binghamton.edu/fridrich/cube.html. 2005.

[5] Carlos A. Hernández. El cubo de Rubik de la A a la Z. http://usuarios.lycos.es/rubikaz/. 2005.

[6] Cristian A. Bravo Lillo. Reino de Menokitán. http://www.menokitan.cl. 2005.

[7] JLD. Cubo de Rubik. http://www.iespana.es/chelis/index.htm. 2005.

[8] Hernán Giraudo. El Cubo de Rubik. http://members.tripod.com/~cubomagico/index.htm. 2005.

[9] Arfur Dogfrey. Introduction to Group Theory. http://members_tripod_com/~dogschool.htm. 2005.

[10] Wikipedia. Cubo de Rubik. http://en.wikipedia.org/wiki/Rubiks_Cube. 2005.

[11] Álvaro Ibáñez. Cómo resolver el Cubo de Rubik. http://www.microsiervos.com. 2005.

[12] Jorge Agz. Cómo resolver el Cubo de Rubik.
http://www.geocities.com/jorge_agz/Rubik0.html. 2005.

[13] Enlexica, Inc. Explore: Rubik's Cube.
http://www.explore-math.com/mathematics/R/Rubik%27s_Cube.html. 2005.