MODELOS DISCRETOS PARA INGENIERÍA

"Cubo Rubik"

Del Salto Gabriel
Quinga Sebastian
Rivera Stephani

NRC: 9901

Sangolquí, 27 de mayo del 2023

Índice

Introducción

La programación es una habilidad esencial en la era digital, y solucionar problemas es una parte fundamental del proceso de programación, estos problemas implican identificar y comprender los obstáculos, diseñar estrategias de solución, implementar y probar soluciones, y finalmente optimizar el código para lograr un rendimiento óptimo. (ConocimientosWeb)

Es por eso que en el presente informe analizaremos las técnicas y herramientas utilizadas para descomponer problemas complejos en partes más manejables, en este caso utilizando también la lógica así como también debemos de tener un pensamiento crítico y creatividad, habilidades totalmente esenciales para resolver el cubo de rubik con matrices en programación. (Domingo Muñoz)

Objetivo General

El objetivo de este informe es diseñar y desarrollar un programa informático que sea capaz de generar y detallar todas las combinaciones posibles del cubo de Rubik 3x3. El programa utilizará algoritmos y técnicas de programación para generar sistemáticamente todas las configuraciones únicas del cubo, brindando una visión completa de las posibles combinaciones y su cantidad.

Objetivo específico

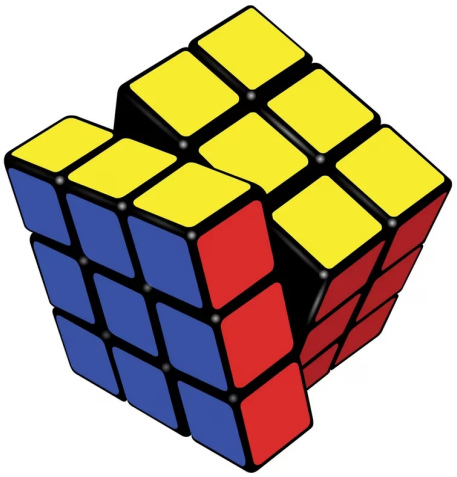
Desarrollar un algoritmo eficiente para generar todas las combinaciones posibles del cubo de Rubik 3x3: El objetivo es diseñar un algoritmo que pueda explorar sistemáticamente todas las configuraciones únicas del cubo, evitando repeticiones y optimizando el proceso de generación. Se buscará minimizar el tiempo y los recursos necesarios para obtener todas las combinaciones.

Validar las combinaciones generadas y verificar su exactitud: Se realizarán pruebas exhaustivas para validar las combinaciones generadas por el programa. Esto implica comprobar que no haya duplicados, que todas las configuraciones sean únicas y que se incluyan todas las posibles combinaciones del cubo de Rubik 3x3. Se verificará la precisión y la integridad de los resultados obtenidos.

Desarrollo

El cubo de Rubik 3x3 es un rompecabezas tridimensional inventado por Ernő Rubik. Está compuesto por un cubo dividido en 6 caras, cada una de un color diferente. El objetivo del juego es mezclar los colores y luego resolverlo para que cada cara esté completamente formada por un solo color.

El cubo de Rubik 3x3 tiene un total de 26 piezas móviles, incluyendo 8 esquinas, 12 aristas y un centro para cada cara. Estas piezas están unidas entre sí mediante un mecanismo de giro interno. Cada cara del cubo puede girarse en sentido horario o antihorario en incrementos de 90 grados.



El cubo de Rubik 3x3 tiene un enorme número de combinaciones posibles. En concreto, hay aproximadamente 43 quintillones. Esto se debe a la cantidad de piezas móviles y movimientos que se pueden realizar. A lo largo de los años, los jugadores y los científicos han desarrollado técnicas y algoritmos avanzados para resolver el cubo de Rubik de manera más eficiente. Esto incluye métodos de capas, métodos de Fridrich, técnicas de bloqueo y más. Estas estrategias ayudan a reducir el número de movimientos necesarios para resolver el cubo y a abordar las diferentes combinaciones posibles.

Combinaciones

El cubo de Rubik tiene un gran número de combinaciones posibles. Para entender cómo se calcula esa cantidad, necesitamos considerar algunas propiedades del cubo.

El cubo de Rubik 3x3x3 tiene 6 caras, cada una con 9 stickers de colores diferentes. En cada cara, los stickers pueden ser permutados entre sí y también pueden ser rotados. Además, cada cara puede rotar alrededor de su centro.

Veamos las distintas propiedades que contribuyen a la cantidad de combinaciones posibles:

Permutación de los stickers: Cada cara del cubo puede tener sus 9 stickers permutados entre sí. Esto significa que hay 9! (factorial de 9) formas diferentes de ordenar los stickers de una cara. Como hay 6 caras, la cantidad total de permutaciones posibles solo para la posición de los stickers es 9!^6.

Rotación de los stickers: Cada sticker en una cara puede rotar en su lugar. Esto proporciona 4 posibles rotaciones para cada sticker. Como hay 9 stickers en cada cara, el número total de combinaciones posibles para las rotaciones de los stickers en una cara es 4^9.

Rotación de las caras: Cada cara puede rotar en incrementos de 90 grados en sentido horario o antihorario. Hay 6 caras, lo que significa que cada cara tiene 4 posibles orientaciones. La cantidad total de combinaciones posibles para las orientaciones de las caras es 4^6.

Multiplicando todas estas propiedades juntas, obtenemos la cantidad total de combinaciones posibles del cubo de Rubik:

Combinaciones totales = (9!^6) * (4^9) * (4^6)

Esta cifra es extremadamente grande, y se estima que hay más de 43 quintillones (4.3 x 10^19) de combinaciones posibles en un cubo de Rubik 3x3x3. Esta es una de las razones por las que resolver un cubo de Rubik es un desafío tan complejo y fascinante.

Explicación fórmula

La fórmula que se utiliza para calcular la cantidad total de combinaciones posibles en un cubo de Rubik 3x3x3 es la siguiente:

Combinaciones totales = (9!^6) * (4^9) * (4^6)

Donde:

9! Es el factorial de 9 (9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1).

^6 indica que este factor se multiplica por sí mismo 6 veces.

4^9 indica que el número 4 se multiplica por sí mismo 9 veces.

4^6 indica que el número 4 se multiplica por sí mismo 6 veces.

Esta fórmula representa la multiplicación de las diferentes propiedades del cubo que mencioné anteriormente: permutación de los stickers, rotación de los stickers y rotación de las caras.

Recuerda que el resultado de esta fórmula es una aproximación de la cantidad total de combinaciones posibles, que se estima en más de 43 quintillones (4.3 x 10^19).

A continuación tenemos el código implementado en el lenguaje Python.

Código:

```
import random

# Crear una matriz 3x3 con valores iniciales
def crear_cara(color, nombre):
    return {"nombre": nombre, "matriz": [[color] * 3 for _ in range(3)]}

# Crear el cubo de Rubik con todas las caras
def crear_cubo():
    cubo = {
      "frontal": crear_cara("⬜", "Frontal"),
       "trasera": crear_cara("🟨", "Trasera"),
       "superior": crear_cara("🟧", "Superior"),
       "inferior": crear_cara("🟥", "Inferior"),
       "derecha": crear_cara("🟩", "Derecha"),
       "izquierda": crear_cara("🟦", "Izquierda")
    }
    return cubo

# Realizar el movimiento R (giro a la derecha de la cara frontal)
def movimiento_r(cubo):
    # Girar la cara frontal
    cubo["frontal"]["matriz"] = list(map(list, zip(*reversed(cubo["frontal"]["matriz"]))))

    # Guardar los elementos afectados por el movimiento
    temp1 = cubo["superior"]["matriz"][0][2]
    temp2 = cubo["superior"]["matriz"][1][2]
    temp3 = cubo["superior"]["matriz"][2][2]
    temp4 = cubo["inferior"]["matriz"][0][2]
    temp5 = cubo["inferior"]["matriz"][1][2]
    temp6 = cubo["inferior"]["matriz"][2][2]

    # Mover los elementos de las caras afectadas
    cubo["superior"]["matriz"][0][2] = cubo["izquierda"]["matriz"][0][2]
    cubo["superior"]["matriz"][1][2] = cubo["izquierda"]["matriz"][1][2]
    cubo["superior"]["matriz"][2][2] = cubo["izquierda"]["matriz"][2][2]

    cubo["inferior"]["matriz"][0][2] = cubo["derecha"]["matriz"][0][2]
    cubo["inferior"]["matriz"][1][2] = cubo["derecha"]["matriz"][1][2]
    cubo["inferior"]["matriz"][2][2] = cubo["derecha"]["matriz"][2][2]

    cubo["derecha"]["matriz"][0][2] = temp1
    cubo["derecha"]["matriz"][1][2] = temp2
    cubo["derecha"]["matriz"][2][2] = temp3

    # Mover la columna del medio sin afectar el elemento del medio
    cubo["izquierda"]["matriz"][0][2] = cubo["trasera"]["matriz"][0][2]
    cubo["izquierda"]["matriz"][1][2] = cubo["trasera"]["matriz"][1][2]
    cubo["izquierda"]["matriz"][2][2] = cubo["trasera"]["matriz"][2][2]

    cubo["trasera"]["matriz"][0][2] = temp4
    cubo["trasera"]["matriz"][1][2] = temp5
```

```
    cubo["trasera"]["matriz"][2][2] = temp6

    return cubo

# Realizar el movimiento F (giro hacia adelante de la cara superior)
def movimiento_f(cubo):
    # Girar la cara superior
    cubo["superior"]["matriz"] = list(map(list, zip(*reversed(cubo["superior"]["matriz"]))))

    # Guardar los elementos afectados por el movimiento
    temp1 = cubo["frontal"]["matriz"][0][0]
    temp2 = cubo["frontal"]["matriz"][1][0]
    temp3 = cubo["frontal"]["matriz"][2][0]

    # Mover los elementos de las caras afectadas
    cubo["frontal"]["matriz"][0][0] = cubo["izquierda"]["matriz"][0][0]
    cubo["frontal"]["matriz"][1][0] = cubo["izquierda"]["matriz"][1][0]
    cubo["frontal"]["matriz"][2][0] = cubo["izquierda"]["matriz"][2][0]

    cubo["izquierda"]["matriz"][0][0] = cubo["trasera"]["matriz"][0][0]
    cubo["izquierda"]["matriz"][1][0] = cubo["trasera"]["matriz"][1][0]
    cubo["izquierda"]["matriz"][2][0] = cubo["trasera"]["matriz"][2][0]

    cubo["trasera"]["matriz"][0][0] = cubo["derecha"]["matriz"][0][0]
    cubo["trasera"]["matriz"][1][0] = cubo["derecha"]["matriz"][1][0]
    cubo["trasera"]["matriz"][2][0] = cubo["derecha"]["matriz"][2][0]

    cubo["derecha"]["matriz"][0][0] = temp1
    cubo["derecha"]["matriz"][1][0] = temp2
    cubo["derecha"]["matriz"][2][0] = temp3

    return cubo

# Realizar el movimiento Rah (giro a la derecha de la cara frontal en sentido antihorario)
def movimiento_rah(cubo):
    # Girar la cara frontal en sentido antihorario
    cubo["frontal"]["matriz"] = list(map(list, zip(*cubo["frontal"]["matriz"])))[::-1]

    # Guardar los elementos afectados por el movimiento
    temp1 = cubo["superior"]["matriz"][0][2]
    temp2 = cubo["superior"]["matriz"][1][2]
    temp3 = cubo["superior"]["matriz"][2][2]
    temp4 = cubo["inferior"]["matriz"][0][2]
    temp5 = cubo["inferior"]["matriz"][1][2]
    temp6 = cubo["inferior"]["matriz"][2][2]

    # Mover los elementos de las caras afectadas
    cubo["superior"]["matriz"][0][2] = cubo["izquierda"]["matriz"][0][2]
    cubo["superior"]["matriz"][1][2] = cubo["izquierda"]["matriz"][1][2]
    cubo["superior"]["matriz"][2][2] = cubo["izquierda"]["matriz"][2][2]

    cubo["inferior"]["matriz"][0][2] = cubo["derecha"]["matriz"][0][2]
```

```
    cubo["inferior"]["matriz"][1][2] = cubo["derecha"]["matriz"][1][2]
    cubo["inferior"]["matriz"][2][2] = cubo["derecha"]["matriz"][2][2]

    cubo["derecha"]["matriz"][0][2] = temp1
    cubo["derecha"]["matriz"][1][2] = temp2
    cubo["derecha"]["matriz"][2][2] = temp3

    # Mover la columna del medio sin afectar el elemento del medio
    cubo["izquierda"]["matriz"][0][2] = cubo["trasera"]["matriz"][0][2]
    cubo["izquierda"]["matriz"][1][2] = cubo["trasera"]["matriz"][1][2]
    cubo["izquierda"]["matriz"][2][2] = cubo["trasera"]["matriz"][2][2]

    cubo["trasera"]["matriz"][0][2] = temp4
    cubo["trasera"]["matriz"][1][2] = temp5
    cubo["trasera"]["matriz"][2][2] = temp6

    return cubo

# Mostrar el cubo en pantalla
def mostrar_cubo(cubo):
    # Imprimir la cara superior
    cara_superior = cubo["superior"]
    print("      ", end="")
    for _ in range(3):
        print("      ", end="")
    for elemento in cara_superior["matriz"][0]:
        print(elemento, end=" ")
    print()

    print("      ", end="")
    for _ in range(3):
        print("      ", end="")
    for elemento in cara_superior["matriz"][1]:
        print(elemento, end=" ")
    print()

    print("      ", end="")
    for _ in range(3):
        print("      ", end="")
    for elemento in cara_superior["matriz"][2]:
        print(elemento, end=" ")
    print()

    print()

 # Imprimir la cara izquierda y la frontal
    cara_derecha = cubo["derecha"]
    cara_frontal = cubo["frontal"]
    cara_izquierda = cubo["izquierda"]
    cara_trasera = cubo["trasera"]
            for fila_derecha, fila_frontal, fila_izquierda, fila_trasera in zip(cara_derecha["matriz"],
cara_frontal["matriz"], cara_izquierda["matriz"], cara_trasera["matriz"]):
```

```
        print("       ", end="")
        for elemento in fila_derecha:
            print(elemento, end=" ")
        print("       ", end="")
        for elemento in fila_frontal:
            print(elemento, end=" ")
        print("       ", end="")
        for elemento in fila_izquierda:
            print(elemento, end=" ")
        print("      ", end="")
        for elemento in fila_trasera:
            print(elemento, end=" ")
        print()


    # Imprimir la cara inferior
    print()
    cara_inferior = cubo["inferior"]
    print("      ", end="")
    for _ in range(3):
        print("      ", end="")
    for elemento in cara_inferior["matriz"][0]:
        print(elemento, end=" ")
    print()

    print("      ", end="")
    for _ in range(3):
        print("      ", end="")
    for elemento in cara_inferior["matriz"][1]:
        print(elemento, end=" ")
    print()

    print("      ", end="")
    for _ in range(3):
        print("      ", end="")
    for elemento in cara_inferior["matriz"][2]:
        print(elemento, end=" ")
    print()

    print()


#  la función desordenar_cubo() para aceptar el número de movimientos
def desordenar_cubo(cubo, movimientos):
    contador_movimientos = 0  # Variable para contar los movimientos realizados
    for _ in range(movimientos):
        movimiento_aleatorio = random.choice([movimiento_r, movimiento_f, movimiento_rah])
        cubo = movimiento_aleatorio(cubo)
        contador_movimientos += 1
    print(f"Se realizaron {contador_movimientos} movimientos al desordenar el cubo.")
    return cubo
```

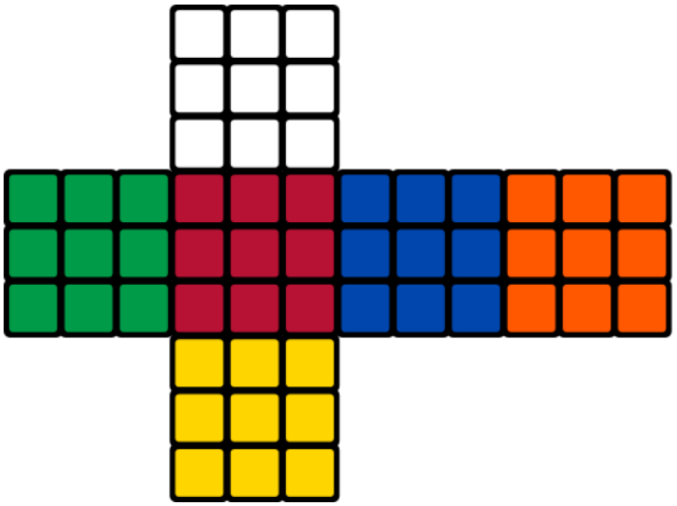
```python
#  la función menu() para que el usuario ingrese lo que desea hacer
def menu():
    cubo = crear_cubo()
    opcion = 0

    while opcion !='4':
        print("\n---------- MENÚ ----------")
        print("1. Mostrar el cubo de Rubik")
        print("2. Desordenar el cubo")
        print("3. Ordenar el cubo")
        print("4. Salir")
        opcion = input("Seleccione una opción: ")

        if opcion == '1':
            mostrar_cubo(cubo)
        elif opcion == '2':
            movimientos = int(input("Ingrese el número de movimientos para desordenar el cubo: "))
            cubo = desordenar_cubo(cubo, movimientos)
            print("\nCubo desordenado:")
            mostrar_cubo(cubo)
        elif opcion == '3':
            cubo = crear_cubo()
            print("\nCubo ordenado:")
            mostrar_cubo(cubo)
        elif opcion == '4':
            print("Usted ha salido del programa, tenga un buen dia!")
        else:
            print("Opción inválida. Intente nuevamente.")

# Ejecutar el menú
menu()
```

En el código anterior nos presenta un cubo rubik en el lenguaje python, nos muestras las caras con la siguiente sintaxis



En el código tenemos los siguientes funciones

- def crear_cara

```
# Crear una matriz 3x3 con valores iniciales
```

- def crear_cubo

```
# Crear el cubo de Rubik con todas las caras
```

- def movimiento_r

```
Realizar el movimiento R (giro a la derecha de la cara frontal)
```

- def movimiento_f

```
# Realizar el movimiento F (giro hacia adelante de la cara superior)
```

- def movimiento_rah

```
# Realizar el movimiento Rah (giro a la derecha de la cara frontal en sentido antihorario)
```

- def mostrar_cubo

```
# Mostrar el cubo en pantalla
```

- def desordenar_cubo

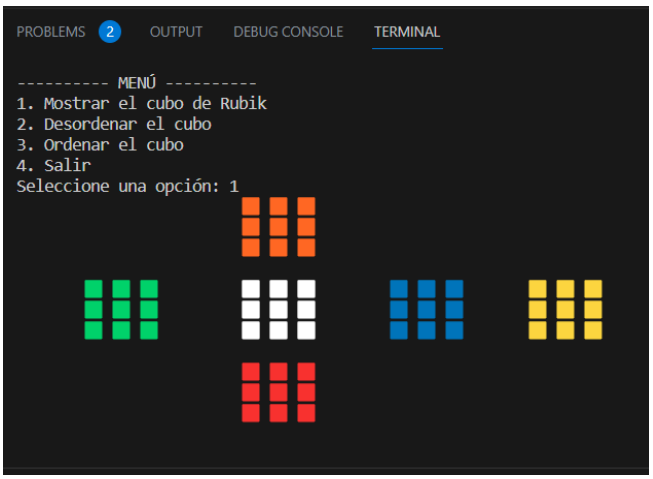
```
# Función desordenar_cubo() para aceptar el número de movimientos
```

- def menu

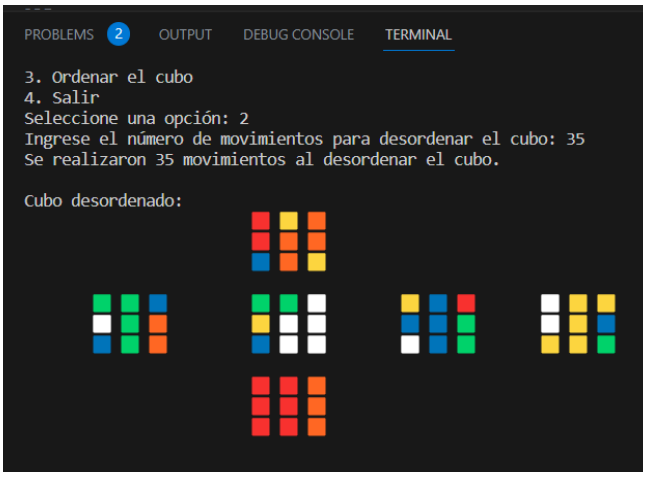
```
# Función menu() para que el usuario ingrese lo que desea hacer
```

En esta opción de menú, tenemos para mostrar por pantalla el cubo ordenado, en la opción número 1, en la opción número 2 es donde se incluyen las funciones de def movimiento_r, def movimiento_f, def movimiento_rah, para desordenar el cubo con los movimientos que ingrese el usuario por pantalla, en la opción número 3 nos muestra el cubo ordenado de nuevo, en la opción número 4 salimos de nuestro programa.

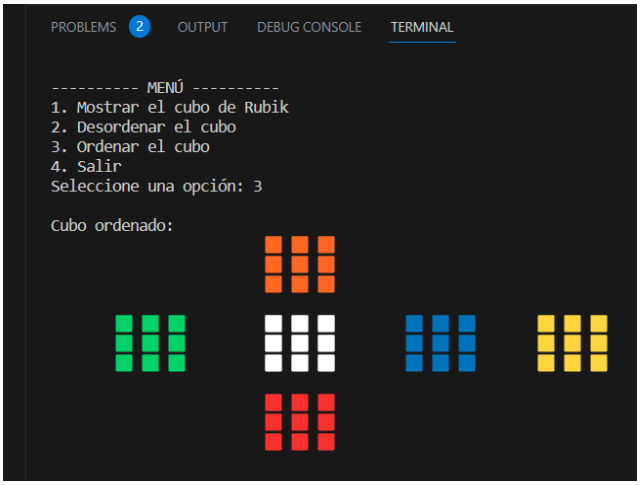
Opción 1:



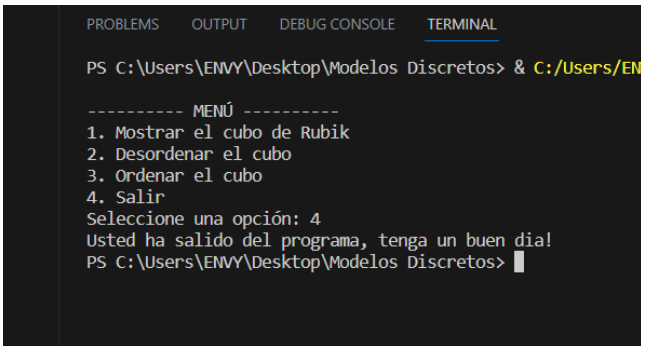
Opción 2:



Opción 3:



Opción 4:



Con esto podemos entender más la lógica con la que se arma el cubo de rubik, desarrollando nuestras destrezas para pasarlo a un lenguaje de programación.

Conclusiones

- El cubo de Rubik tiene un número masivo de combinaciones posibles, estimado en más de 43 trillones. Esto muestra la complejidad inherente del rompecabezas y la diversidad de desafíos que se pueden encontrar al intentar resolverlo.
- Para resolver el cubo de Rubik, se utilizan algoritmos específicos que manipulan las piezas del cubo de manera predefinida. Estos algoritmos permiten realizar movimientos específicos para cambiar las piezas de lugar y restaurar el cubo a su estado resuelto. La comprensión y aplicación de estos algoritmos es fundamental para resolver el cubo de Rubik de manera eficiente.
- La resolución del cubo de Rubik mediante programación implica desarrollar algoritmos y técnicas para manipular y reorganizar las piezas del cubo. Estos algoritmos se implementan utilizando lenguajes de programación y se ejecutan en un software o un robot que interactúa físicamente con el cubo.

Recomendaciones

- Aprende los algoritmos básicos: Para resolver el cubo de Rubik de manera eficiente, es importante familiarizarse con los algoritmos básicos que se utilizan en su resolución. Estos algoritmos están compuestos por una serie de movimientos que manipulan las piezas del cubo de una manera predefinida. Puedes encontrar tutoriales y recursos en línea que te enseñarán estos algoritmos paso a paso.
- Práctica la resolución manualmente: Antes de aventurarte en la programación, es recomendable que practiques la resolución del cubo de Rubik de forma manual. Esto te ayudará a comprender los movimientos y algoritmos básicos, así como a desarrollar habilidades de manipulación y visualización espacial. Puedes encontrar guías impresas o tutoriales en línea para seguir paso a paso.
- Explora soluciones programáticas: Una vez que te sientas cómodo con la resolución manual, puedes explorar la resolución del cubo de Rubik mediante programación. Esto implica desarrollar algoritmos y técnicas para manipular y reorganizar las piezas del cubo utilizando lenguajes de programación. Puedes comenzar por implementar los algoritmos que ya has aprendido en un software o utilizar bibliotecas específicas de cubo de Rubik disponibles en diferentes lenguajes de programación.

Bibliografía

ConocimientosWeb. “Programación: Pasos para la solución de problemas.” *Programación: Pasos para la solución de problemas*, 09 02 2023, https://conocimientosweb.net/zip/article814.html. Accessed 25 06 2023.

Domingo Muñoz, Jose. “Resolución de problemas.” *Resolución de problemas*, PLEDIN 3.0, 10 01 2023, https://plataforma.josedomingo.org/pledin/cursos/programacion/curso/u01/. Accessed 25 06 2023.