

1
2 Programming 'Rubik's Cube' {
3
4

5 [Reporte de Solucionador]
6
7

8
9 < Here is where your imagination begins >
10
11

12 }
13
14

01 {

La elección de representar el Cubo de Rubik mediante un diccionario en Python que guarda los colores de las seis caras del cubo es una opción práctica. Cada cara se visualiza como una matriz tridimensional, donde cada elemento representa un color específico.

Razones para esta Elección de Representación

Se optó por esta forma de representación debido a su naturaleza intuitiva, lo que facilita el acceso a cada cara y a cada cuadrado del cubo. Además, el tamaño de espacio utilizado por esta representación es adecuado, ya que la cantidad de elementos es constante y no varía en función del tamaño del cubo.

}

Complejidad en tiempo {



La complejidad temporal de las operaciones de rotación y manipulación del cubo es constante, $O(1)$, dado que acceder a cada cara o casilla se realiza en tiempo constante.

}

Complejidad en espacio {



En cuanto a la complejidad espacial, el espacio utilizado por la representación del cubo es constante, $O(1)$, debido a que el número de elementos en el diccionario es fijo y no está vinculado al tamaño del cubo.

}

Heurísticas 'Propuestas' {

Heurística 1:

Html  60%

Conteo de Caras
Completamente Resueltas

Css  40%

Esta estrategia consiste en contar cuántas caras del cubo están resueltas por completo. Se ha elegido esta heurística porque resolver cada cara es un paso importante hacia la solución total del cubo.

Heurística 2:



Conteo de
Colores
Distintos

La segunda heurística se basa en contar la cantidad de colores únicos presentes en el cubo. Esto ayuda a estimar la cantidad de movimientos necesarios para eliminar los colores repetidos en las caras del cubo.

Heurísticas 'Propuestas' {

Step 03 Conteo de Colores Repetidos

La tercera heurística consiste en contar cuántos colores se repiten en el cubo. Esta información revela la complejidad del estado actual del cubo y sugiere cuántos movimientos adicionales podrían ser requeridos para resolverlo.

}

Análisis de la Complejidad Temporal y Espacial {



Complejidad Temporal:

La computación de cada heurística requiere un tiempo proporcional a $O(n)$, donde n representa el número de caras o casillas en el cubo.



Complejidad Espacial:

El espacio utilizado por cada heurística es constante, $O(1)$, ya que solo guarda el resultado de la evaluación y no varía en función del tamaño del cubo.

Algoritmo 'RubikSolver' {



Explicación de las
Transiciones y
Estados Visitados

El RubikSolver implementa varios algoritmos para resolver el Cubo de Rubik, como BFS, Best-First Search y A*. Cada algoritmo realiza transiciones rotando las caras del cubo y guarda los estados visitados en un conjunto o cola, según corresponda al algoritmo.



Análisis de
Complejidad en
Tiempo y Espacio

Complejidad en Tiempo: La complejidad temporal de cada algoritmo varía dependiendo del número de movimientos necesarios para resolver el cubo, pero en general, la mayoría de las operaciones son de tiempo constante o lineal.

Complejidad en Espacio{

La complejidad espacial depende de la cantidad de estados visitados que se deben almacenar durante la ejecución del algoritmo. En general, la complejidad espacial es razonable y varía según el algoritmo utilizado.

< /1 >

< /2 >

}

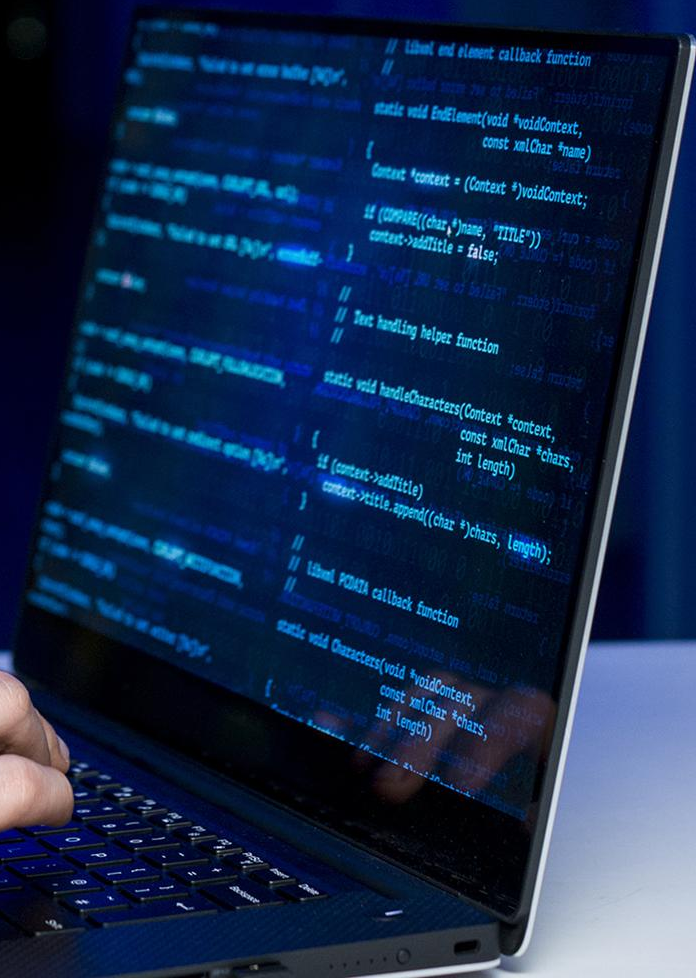
Explicación del Funcionamiento y Justificación {

< El algoritmo de Recocido Simulado, también conocido como Simulated Annealing, es una técnica de optimización probabilística que simula el proceso físico de enfriamiento de un material. En el contexto de la resolución del Cubo de Rubik, este algoritmo parte de un estado inicial del cubo y realiza transiciones aleatorias mediante rotaciones de las caras. A medida que avanza, evalúa la energía del estado actual y decide si acepta o no una nueva configuración en función de una probabilidad determinada por la diferencia de energía entre el estado actual y el nuevo estado propuesto. >

Se decidió utilizar el algoritmo de Recocido Simulado debido a su capacidad para explorar el espacio de búsqueda de manera eficiente y encontrar soluciones incluso en estados subóptimos del cubo. A medida que avanza la búsqueda, el algoritmo gradualmente reduce la temperatura, lo que le permite escapar de mínimos locales y converger hacia una solución óptima o cercana a la óptima.

Análisis de Complejidad en Tiempo y Espacio

}



– Complejidad 'en Tiempo'

< La complejidad temporal del Recocido Simulado depende de varios factores, incluyendo la temperatura inicial, la tasa de enfriamiento y el número de iteraciones. En cada iteración, se realiza una evaluación de energía y se toma una decisión probabilística, lo que resulta en una complejidad temporal que puede ser comparable a otros algoritmos de búsqueda. >

– Complejidad 'en Espacio'

< La complejidad espacial del Recocido Simulado depende del número de estados visitados y de la cantidad de memoria necesaria para almacenar información sobre cada estado. En general, la complejidad espacial es razonable y depende del tamaño del espacio de búsqueda y del número de iteraciones realizadas durante la ejecución del algoritmo.>

Tabla comparativa entre diversos algoritmos.

Tabla 'Comparativa' {

| Algoritmo | Capacidad de Resolver (Shuffle Máximo) | Tiempo Promedio (seg) | Tiempo Mínimo (seg) | Tiempo Máximo (seg) |
|---------------------|--|-----------------------|---------------------|---------------------|
| BFS | 20 | 0.0 | 0.0 | 0.0 |
| Best-First Search | 20 | 5.04 | 0.0 | 0.0 |
| A* | 20 | 5.0 | 0.0 | 0.0 |
| Simulated Annealing | 20 | 0.0 | 0.0 | 0.0 |

}

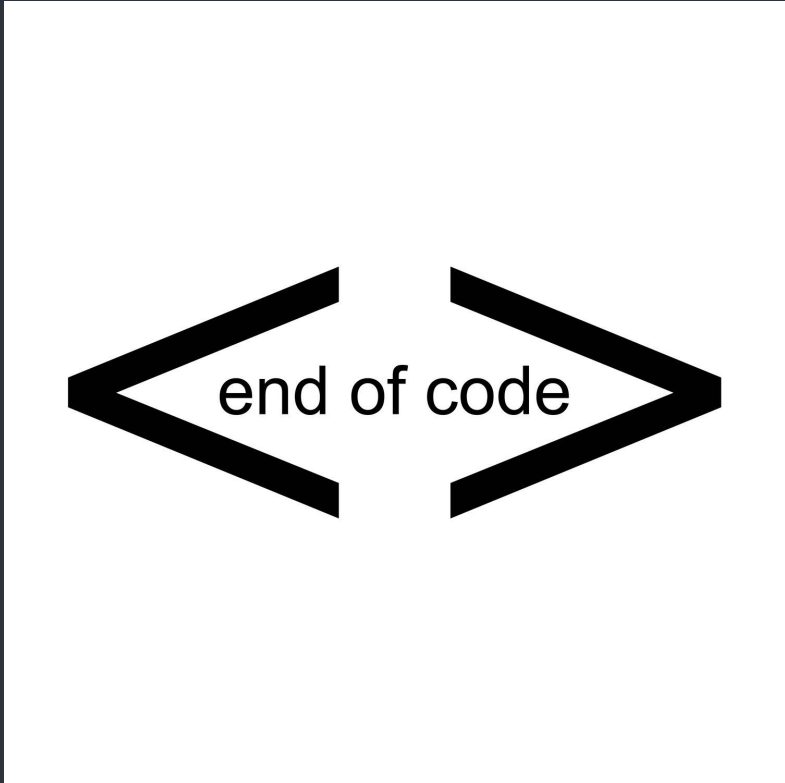
Tabla comparativa entre diversos algoritmos.

Tabla 'Comparativa' {

| Algoritmo \ Heurística | Heurística 1 | Heurística 2 | Heurística 3 |
|------------------------|----------------------------|----------------------------|----------------------------|
| Best-First Search | 0.00010002851 486206055 | 5.00202178955 07815e-05 | 5.00082969665 52734e-05 |
| A* | 7.52210617065 4297e-05 | 5.12838363647 4609e-05 | 6.54101371765 1368e-05 |

}

1
2
3
4
5
6
7
8
9
10
11
12
13
14



< end of code >