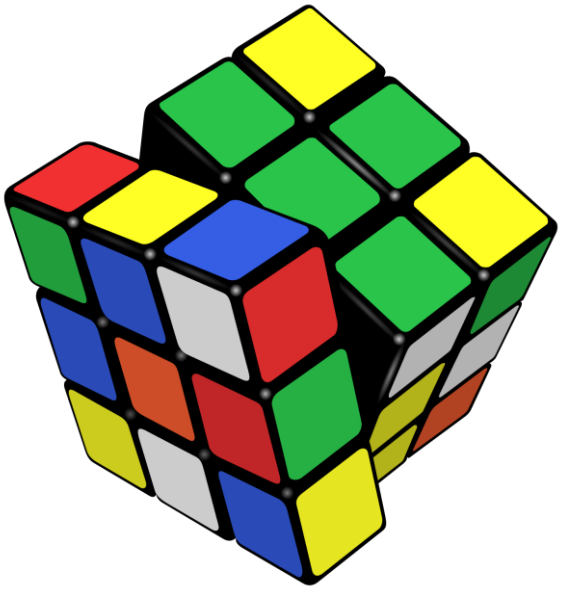
IMPLEMENTACIÓN DE UN ARTEFACTO SOFTWARE N-CUBO DE RUBICK



11/11/2019

Andrés González Díaz
Estudiante de Ingeniería Informática. ESI. UCLM
andres.gonzalez8@alu.uclm.es

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

ÍNDICE

0. INTRODUCCIÓN

Este documento recoge la memoria técnica de la resolución del cubo de Rubik de tamaño N. El contenido de esta memoria recopila el proyecto del laboratorio de Sistemas Inteligentes.

Además, para coordinarnos y gestionar mejor las distintas versiones y actualizaciones del código mediante el uso de GIT y GitHub, una plataforma de desarrollo colaborativo de software para alojar proyectos.

Para el almacenamiento y el trabajo en conjunto del código de esta práctica queda reflejado en el siguiente repositorio de GitHub:

https://github.com/Ofeucor/Rubick

1. ANALISIS

1.1 Definición del problema

El problema planteado como proyecto de laboratorio consiste en realizar un programa capaz de encontrar una solución a un cubo de Rubik de tamaño NxNxN. El programa debe ser capaz de simular todos los movimientos posibles, de cualquier cubo y de usar el formato de archivos ". json" para leer y escribir el objeto "Cube". Además para facilitar el análisis de los movimientos y del cubo se ha incorporado una forma de representación del cubo.

Como objetivos de la tarea 0 tendremos que realizar una representación interna para el cubo, lectura del objeto cubo original a partir de un archivo ".json" y la generación de todos los posibles movimientos.

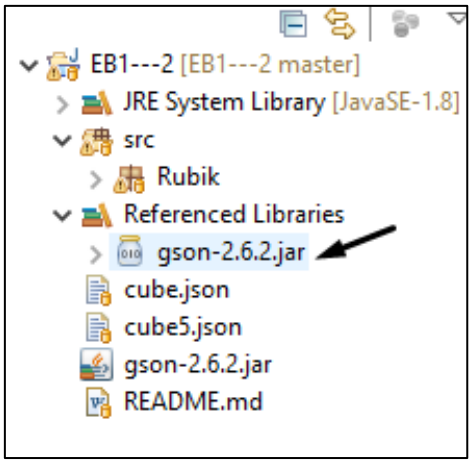
1.2 Solución al problema

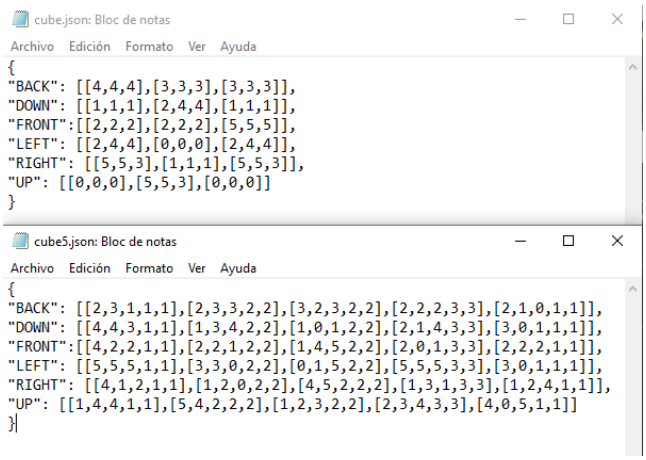
El lenguaje de programación que hemos elegido es Java y el entorno de desarrollo Eclipse.

Como hemos mencionado anteriormente, deberemos resolver un cubo de Rubik con las dimensiones que se deseen, para ello hemos implementado distintas clases:

- Cube. Clase encargada de la representación interna del cubo.

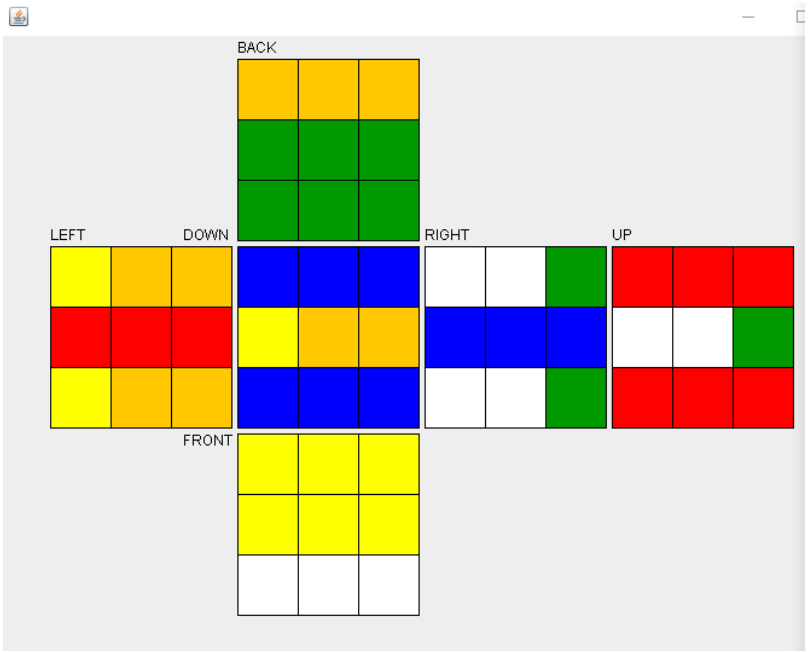
- Reader. Clase encargada de la lectura y escritura de archivos “. json”, para ello hacemos uso de la librería de GSON para Java de Google para leer y tratar archivos “. json”.





- Resolver. Clase principal o main () que lleva al cabo las distintas operaciones elegidas por el usuario o que pone en práctica una de las formas de búsqueda estudiadas en las clases de teoría.

- Window. Clase que muestra el cubo de manera gráfica y visual.



Tras realizar estas clases y comprobar que se realiza correctamente la lectura del cubo de Rubik y los movimientos de manera correcta, mediante cubos de NxNxN para comprobar la correcta implementación del código.

GitHub: Ofeucor

2. DISEÑO

2.1 Estructuras propuestas

Clase Cube.java:

Clase encargada de la representación interna del cubo que cuenta con 6 matrices de tipo int para representar las distintas caras del cubo (BACK, LEFT, DOWN, RIGHT, UP y FRONT), un atributo (int) id que servirá como identificador y como atributo extra un Cube que hace referencia al cubo origen del que proviene, guardando una copia del cubo previo a este tras realizar cualquier movimiento.

Clase Window.java:

Clase encargada de la representación gráfica del cubo, para ello hacemos uso del cubo que vamos a representar y de las librerías java.awt.Graphics, javax.swing.JFrame y javax.swing.JPanel para la creación del panel y del gráfico.

Clase Reader.java:

Clase encargada de la lectura y escritura de archivos ".json", para poder realizar estas operaciones hacemos uso de la librería mencionada anteriormente "gson".

Clase Frontera.java:

Clase encargada de la representación de la frontera en la resolución de un grafo, una lista ordenada ascendentemente en función de "f", que contiene los siguientes métodos: Frontera() que crea una frontera vacía, insertar() añade un nuevo nodo a la frontera, estaVacia() indica si la frontera está vacía o no, peek() coge y saca el primer elemento de la frontera, first() que recupera el primer elemento sin eliminarlo, y los métodos de ordenación de flotar() y hundir() .

Clase Nodo.java:

Clase representativa de los nodos de un árbol que contiene la información del nodo padre y del dominio; estado actual, costo de camino realizado, acción realizada para llegar al estado actual, profundidad y "f" como el valor que determina la posición en la frontera que se generara de forma aleatoria con números entre [0,50].

Clase Estado .java:

Clase que recoge la representación del cubo y la función objetivo que determina si un estado es objetivo o no, es decir, el cubo está resuelto.

.

Clase Main.java:

Clase principal que recoge y hace uso de las distintas estructuras mencionadas para ofrecer una solución. Para esta primera parte del proyecto se ha planteado un menú simple por consola para comprobar los distintos tipos de movimiento y un método sucesores() que realiza y gestiona todas las posibles acciones. El control de errores realizado por el programa es mínimo ya que aún no hemos tenido oportunidad ni el tiempo de implementar las distintas medidas.

Esta clase recoge además los movimientos que se pueden hacer desde cada cubo, recogidos en tres métodos moveLX(), moveBX() y moveDX(), a los que se les pasa como argumentos tipo de rotación(positiva o negativa), columna o fila sobre la que se aplica y el cubo al que se le aplica el tipo de movimiento.

2.2 Algoritmos de búsqueda

Para la resolución del cubo de rubik se ha implementado los distintos tipos de búsqueda diferenciando entre informada y no informada. El algoritmo desarrollado tiene como entrada la profundidad máxima y su estado inicial.

2.2.1 Búsqueda no informada

Las búsquedas no informadas desarrolladas son:

- Búsqueda en profundidad: Es un algoritmo para recorrer o buscar elementos de un grafo que consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.
- Búsqueda en anchura: Es un algoritmo para recorrer o buscar elementos de un grafo que comienza por la raíz y se explora todos los hijos de este nodo. A continuación, se explora cada uno de los hijos de los hermanos y así sucesivamente hasta encontrar la solución.
- Búsqueda en coste uniforme: Es un algoritmo para recorrer o buscar elementos de un grafo que consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto definido por el coste de cada acción.

2.2.2 Búsqueda informada

Las búsquedas informadas desarrolladas son:

- Búsqueda Voraz:
- Búsqueda A*:

GitHub: Ofeucor

ANEXO I

Library GSon - gson-2.6.2.jar