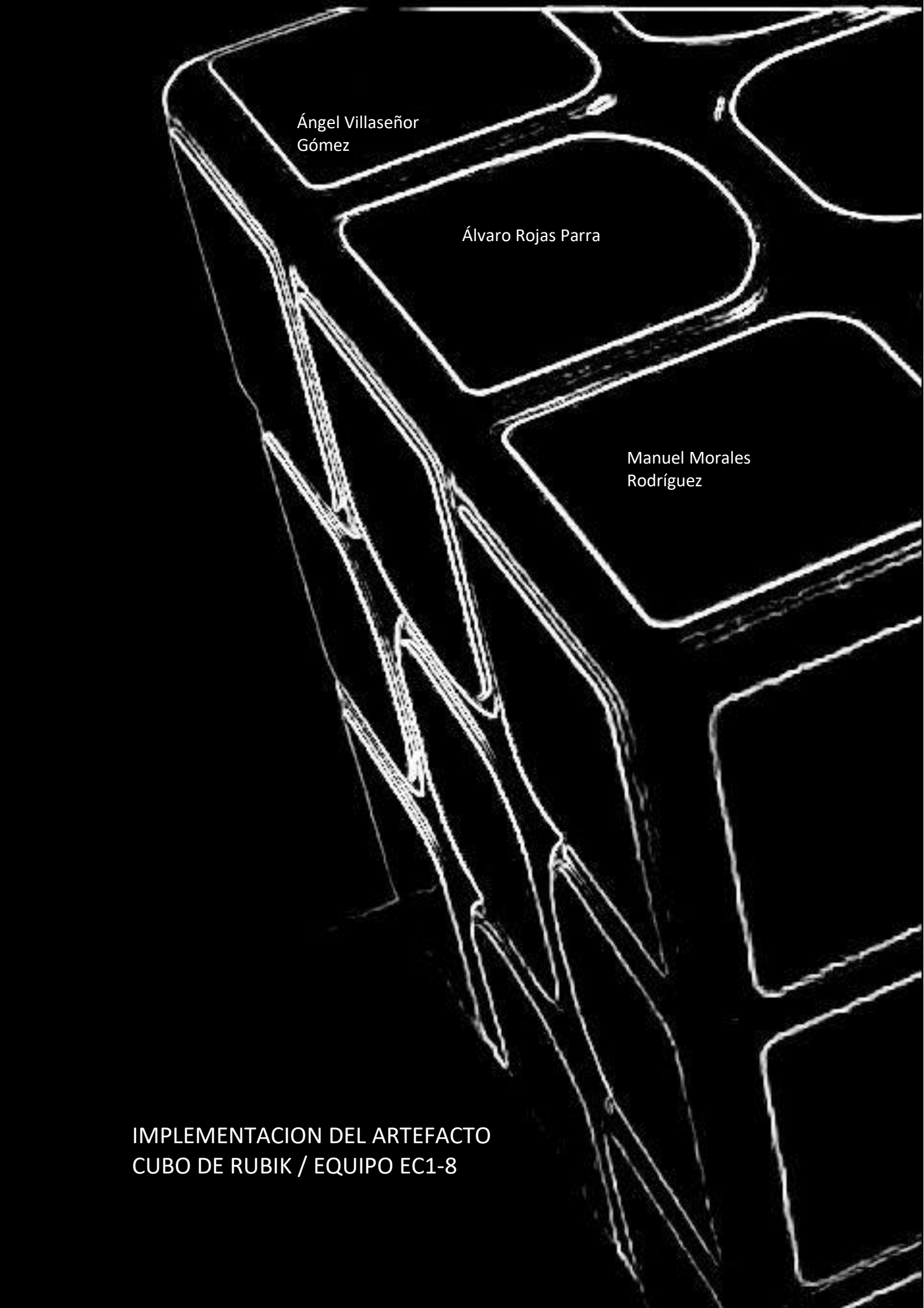


Ángel Villaseñor Gómez

Álvaro Rojas Parra

Manuel Morales Rodríguez

IMPLEMENTACION DEL ARTEFACTO CUBO DE RUBIK / EQUIPO EC1-8

INDICE

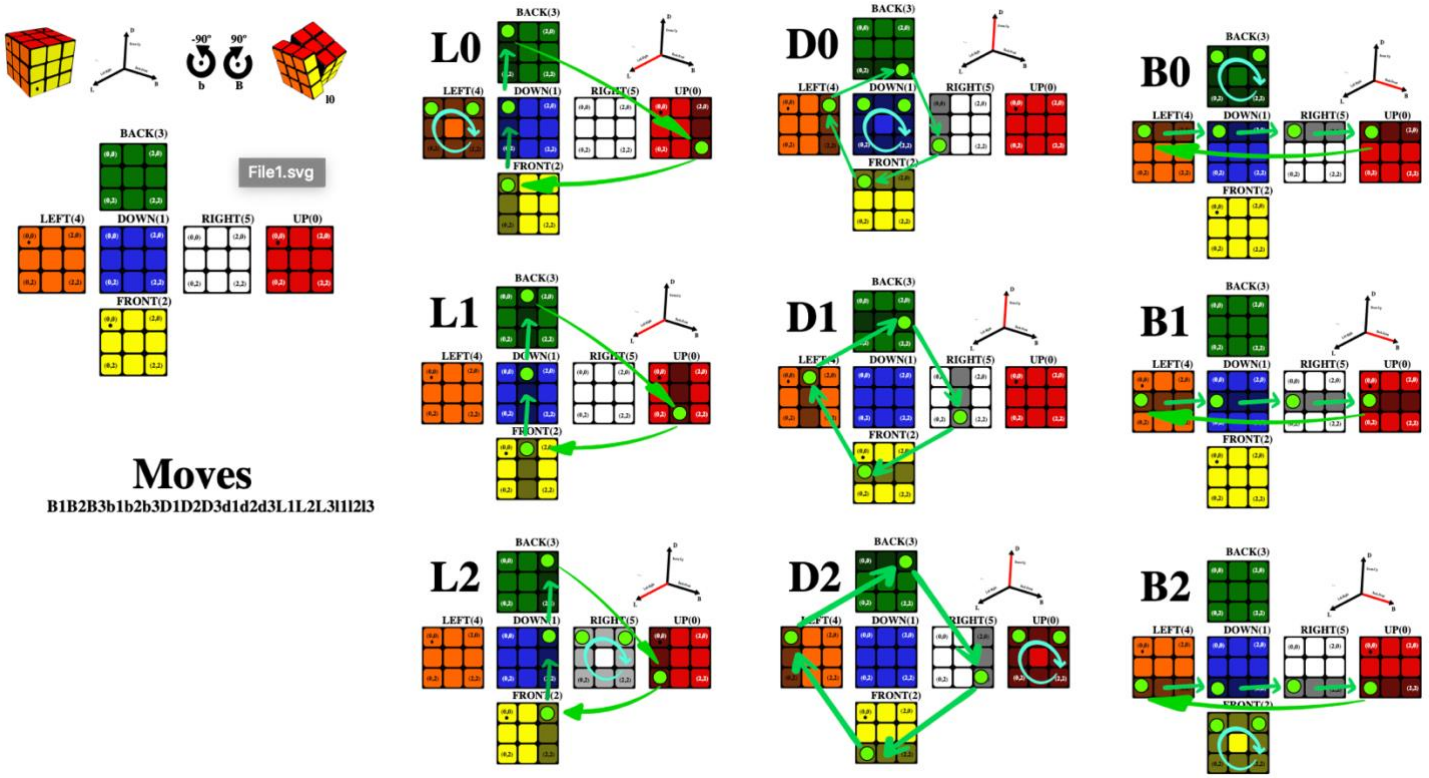
CONTEXTO DEL EJERCICIO

El objetivo de la práctica es implementar un programa para resolver un cubo de rubic con dimensiones NxNxN donde N podrá ser cualquier número entero positivo, usando las técnicas de búsqueda estudiadas en la asignatura.

Para ello, debemos construir el artefacto a partir de un fichero JSON. Debemos poder implementar primero todos los posibles movimientos válidos *(se adjunta foto),* y después aplicar los diferentes algoritmos de búsqueda.

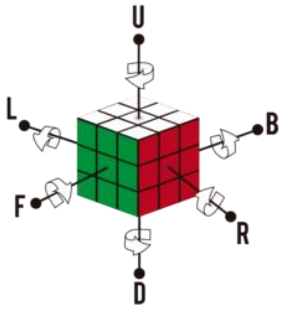
También debemos crear un repositorio en la plataforma GitHub para ir subiendo el código fuente de nuestro proyecto y llevar un control de versiones.

El lenguaje que se ha utilizado para implementar cada hito y por tanto toda la practica ha sido JAVA y los entornos de desarrollo utilizados han sido Visual Studio Code y NetBeans.



DEFINICIÓN DE HITOS

HITO. 0



Los objetivos para este hito son:

- Elegir un lenguaje de programación.
- Crear un repositorio en GitHub.
- Generar el cubo a partir de un archivo JSON.
- Clonar el objeto cubo en memoria.
- Generar los movimientos posibles del cubo y actualizar el cubo tras aplicar un movimiento.

El lenguaje de programación elegido fue JAVA, ya que todos los miembros del equipo estábamos muy familiarizados con ese lenguaje al haberlo utilizado durante toda la carrera.

El repositorio de GitHub fue creado tal y como se explicó en la primera sesión, incluyendo al profesor como colaborador del mismo.

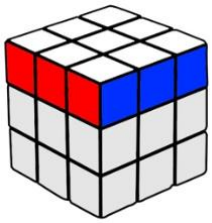
El cubo ha sido creado obteniendo primero el valor de N del archivo JSON que se pasa al programa para posteriormente crear cada cara como una matriz de NxN filas y columnas, es decir, hemos considerado cada cara como una matriz para poder trabajar mejor con cada cara y evitar confusiones o complicaciones. Una vez que todas las matrices han sido creadas, se procede a rellenar cada una de las mismas con el estado inicial del cubo.

Adicionalmente, para identificar cada celda, hemos creado un método en el cual se recorre el cubo entero y se le asigna un ID a cada celda.

Para clonar el objeto cubo, se crea un nuevo objeto y se recorre el que ya existía previamente para copiarlo entero al nuevo objeto. Para ello hemos utilizado la librería Cloneable de JAVA y hemos copiado cara a cara del cubo existente al nuevo.

Para generar los movimientos de cada cubo se ha creado una matriz auxiliar para poder sincronizar todas las caras una vez que se modifica una de ellas, pasando celdas de una fila a otra recorriendo y sustituyendo cada celda por su correspondiente valor nuevo.
Para realizar la rotación del cubo simplemente se copia el estado de las matrices a su nueva posición tras rotar el cubo.
De esta manera nos aseguramos también de que el cubo se va actualizando cada vez que se aplica un movimiento.

HITO. 1



Los objetivos para este hito son:

- Definir el espacio de estados.
- Definir el problema.
- Definir el nodo del árbol de búsqueda.
- Definir la frontera del árbol de búsqueda.

El espacio de estados lo hemos implementado de manera que primero se obtiene el estado inicial del cubo a través del archivo JSON y a partir de ese estado, se comprueba qué acciones se pueden realizar de manera posterior y se almacenan en una lista de Acciones.
Para cada una de estas acciones, se crea un estado sucesor con la acción(movimiento) que hay que llevar a cabo para alcanzarlo junto al nuevo estado del cubo y su coste para finalmente almacenar todos los estados sucesores en una lista de sucesores del estado inicial.

Para definir el problema, se ha definido un getter del estado inicial y a partir de ese estado inicial, se obtiene su espacio de estados, el cual estará formado por ese estado inicial y sus sucesores los cuales se obtienen de la manera explicada previamente.

Para comprobar que el estado del cubo es el estado objetivo, el cual no es otro que el estado en que el cubo está resuelto, se recorren las caras del cubo y se comprueba que cada una de las celdas de las caras tengan el mismo valor, es decir, que sean del mismo color. En el momento en que hay una sola celda de diferente color respecto al resto de celdas de la misma cara, el estado del cubo no es el estado objetivo.

El nodo del árbol ha sido creado mediante el almacenamiento de todos sus atributos (nodo padre, estado, coste, acción, profundidad y función) en un objeto Nodo con sus correspondientes getters y setters para poder acceder a cada uno de estos atributos y su información.

Para hacer la frontera, inicialmente la realizamos con tres estructuras de datos: PriorityQueue, Pila y ArrayList, porque preferíamos tener varias opciones a elegir para cuando tocase hacer el árbol de búsqueda, en función de cual nos daba mas facilidades, elegir una de esas tres.
Finalmente, en el hito 2 a la hora de realizar el árbol de búsqueda elegimos la PriorityQueue ya que al ser una cola de prioridades que nos permitía almacenar los nodos del árbol con mucha facilidad ya que estos se ordenan por prioridades de manera ‘automatica’ debido a la naturaleza de la cola. Mediante esta cola de prioridades podemos añadir nodos, eliminarlos o comprobar si está vacía la cola.

HITO. 2

Los objetivos para este hito son:

- Definir el algoritmo de búsqueda en anchura.
- Definir el algoritmo de búsqueda en profundidad simple, acotada e iterativa.
- Definir el algoritmo de búsqueda en costo uniforme.

La implementación de los algoritmos de búsqueda es similar en cierta manera, ya que primeramente, dado un problema y una profundidad máxima, se obtiene el nodo inicial a partir del problema y se crea su frontera. Después, mientras que no se encuentra una solución, el programa se sigue ejecutando y por tanto se obtiene el espacio de estados y los sucesores a partir del estado en que se encuentra el cubo, (utilizando todo lo implementado en los hitos previos) y se crean los nodos del árbol de búsqueda, asignándoles su ID e incluyéndolos en la frontera y en caso de visitarlo, incluirlo en la lista de nodos visitados.

La forma de crear los nodos del árbol dependerá obviamente del algoritmo de búsqueda elegido, para ello hemos usado un switch en un mismo método para que, en función del algoritmo, los nodos se generen de una u otra forma.

Esto se aplica para todos los algoritmos de búsqueda excepto para el algoritmo de profundidad iterativa el cual hemos implementado aparte ya que tiene la particularidad de que es necesario especificar por parte del usuario que usa el programa, el incremento de la profundidad, mientras que el resto de algoritmos solo necesitan el valor de la profundidad máxima y al ser algo común a todos los algoritmos por simpleza y comodidad preferimos implementarlos todos en un mismo método a través del switch mencionado.

Así pues, para todos estos algoritmos 'comunes', a través del switch se generan los nodos correspondientes al árbol de búsqueda y de manera correspondida a la f(n) del algoritmo que en caso de ser en anchura será la profundidad del nodo, en caso de ser costo uniforme será el coste acumulado para llegar al nodo… etc.

Por otro lado, el algoritmo de profundidad iterativa se ha implementado en un método (Busqueda), por la razón explicada previamente y por si solo no hace el algoritmo, es decir, en el método 'Busqueda' se comprueba si hay una solución en la profundidad máxima y en caso de no haberla y en caso de que se pueda incrementar la profundidad sin pasar la profundidad máxima, esta se va aumentando y llama al método Búsqueda_Acotada para que ahora si, se ejecute el algoritmo.

Por último, cuando se ha encontrado la solución, en lugar de crear un método creaSolucion como se especifica en las diapositivas, debido a que consideramos el código para obtener la solución bastante simple preferimos hacerlo directamente realizando un recorrido partiendo del nodo solución hasta llegar al nodo raíz para

almacenar todos los nodos que forman parte de la solución en una LinkedList. Decidimos usar una LinkedList ya que así se nos permite enlazar el nodo padre con el nodo hijo desde el nodo raíz pasando por los nodos intermedios hasta llegar al nodo solución.

Este camino solución se almacena en un fichero de texto de salida siempre y cuando exista una solución, de lo contrario no se generará el fichero. En ese fichero, aparecerán datos como el número de nodos generados, la profundidad y el coste de la solución y posteriormente saldrá, para cada nodo del camino de solución, su ID, su coste, su f(n), su profundidad… etc.

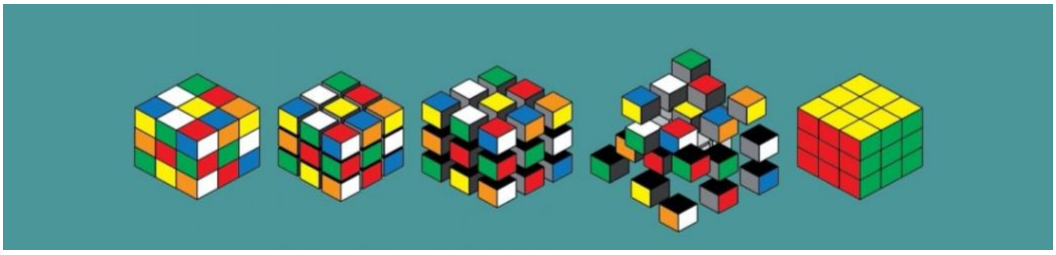
HITO. 3

Los objetivos para este hito son:

- Definir el algoritmo de búsqueda A*.
- Definir el algoritmo de búsqueda Voraz.
- Definir la heurística del cubo.

El algoritmo de búsqueda A* y Voraz han sido implementados como el resto de los algoritmos del hito 2, ya que al igual que en el resto de los casos exceptuando el de profundidad iterativa, solo necesitan la profundidad máxima para comenzar a trabajar, la única diferencia es, otra vez, la f(n) que en el caso de A* será la suma de la heurística y el costo acumulado mientras que en Voraz, será la heurística.

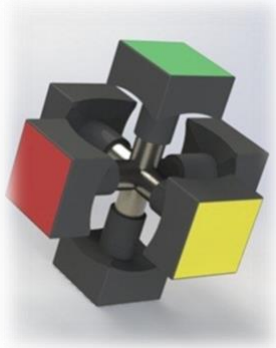
Sin embargo, surge el problema de que antes de implementar el algoritmo A* y Voraz, es necesaria una forma de asignar las heurísticas. Para ello, nos hemos fijado en la ayuda que se proporciona por parte del profesor para entender e implementar la forma de asignar la heurística, primero calculando la entropía calculando las celdas de cada color de cada cara y finalmente haciendo el calculo matemático indicado por el profesor.



MANUAL DE USUARIO

Al lanzar nuestro programa desde la clase “cuboderubic” aparecerá un menú con las distintas opciones de búsqueda que existen para encontrar la solución:

```
1. Anchura
2. Profundidad simple
3. Profundidad Acotada
4. Profundidad Iterativa
5. Coste Uniforme
6. Voraz
7. A*

(Selecciona una busqueda):
```



A continuación, se selecciona el algoritmo de búsqueda que se desea, introduciendo su número de opción y se pulsa ENTER.

```
(Selecciona una busqueda): 1
Dame la profundida máxima:
```

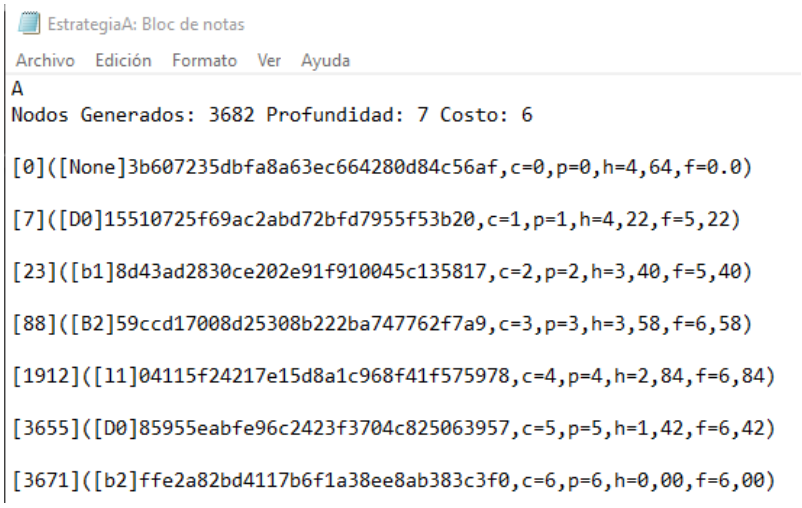
Posteriormente, saldrá otro menú para que seleccionemos la profundidad máxima de la búsqueda de la solución. Se introduce la profundidad deseada y al pulsar ENTER el programa comenzará a trabajar.

Si seleccionamos la opción 4. Profundidad iterativa, además del menú de profundidad máxima, se despliega un menú exclusivo de ese algoritmo de búsqueda para seleccionar el incremento de profundidad que deseemos. Una vez elegido, al pulsar ENTER el programa comienza a buscar la solución.

```
1. Anchura
2. Profundidad simple
3. Profundidad Acotada
4. Profundidad Iterativa
5. Coste Uniforme
6. Voraz
7. A*

(Selecciona una busqueda): 4
Dame la profundida máxima: 3
Dame el incremento de Profundidad Iterativa:(por defecto 1)
```

Cuando el programa ha encontrado una solución, informa de que se ha encontrado una solución y posteriormente se genera un fichero con la solución.

```
run:
1. Anchura
2. Profundidad simple
3. Profundidad Acotada
4. Profundidad Iterativa
5. Coste Uniforme
6. Voraz
7. A*

(Selecciona una busqueda): 7
Dame la profundida máxima: 6

FICHERO GENERADO
BUILD SUCCESSFUL (total time: 6 seconds)
```



Si el programa no es capaz de encontrar la solución lo mostrara con un mensaje por pantalla como el siguiente

```
(Selecciona una busqueda): 3
Dame la profundida máxima: 4
No se ha podido encontrar solucion
BUILD SUCCESSFUL (total time: 6 seconds)
```

Si se desea abortar la ejecución del programa, pulsando la combinación de teclas CTRL+C es posible.

```
(Selecciona una busqueda): 4
Dame la profundida máxima: 19
Dame el incremento de Profundidad Iterativa:(por defecto 1) 5
BUILD STOPPED (total time: 37 seconds)
```

OPINIÓN PERSONAL DE CADA MIEMBRO

Álvaro Rojas Parra:

En primer lugar, la práctica me resulta llevadera porque todos los conceptos relacionados con los árboles de búsqueda se explican con mucha antelación en las clases teóricas y a la hora de hacer la práctica todo resulta mas familiar y conocido que si se hiciese una práctica de algo de lo que no se tiene ninguna o casi ninguna noción.

Por mi parte la practica ha sido más fácil de realizar gracias a eso, ya que primero se explica en la teoría todo lo relacionado con los árboles de búsqueda y cuando ya se ha hecho el examen y el trabajo teórico, se realiza la entrega final y el posterior examen, es decir, las entregas de la práctica están muy bien sincronizadas con el resto de apartados de la asignatura, facilitando todo.

También es llevadera porque en las sesiones prácticas, se explica bien cuál debe ser el funcionamiento del cubo de Rubick y consejos para llevar a cabo la implementación. También es un acierto el hecho de que cada equipo pueda desarrollar su código en el lenguaje de programación que desee y sin ninguna restricción de uso de diferentes estructuras o librerías.

También se agradece que el profesor, después de cada entrega se interese por saber el estado del proyecto de cada equipo y proporcione consejos para mejorar el proyecto o sugerencias para realizar las entregas futuras.

Por otro lado, también es interesante porque en mi opinión el cubo de Rubick es un 'problema' que mucha gente conoce pero que no tantos saben resolverlo. En mi opinión esta práctica ayuda a saber cómo resolverlo o por lo menos entender mejor como se podría resolver gracias al uso de los diferentes algoritmos de búsqueda que se implementan en la práctica.

Ángel Villaseñor Gómez

El objetivo de la practica estaba claro desde el principio y no es otro que el de aprender las diferentes estrategias de búsqueda. Al principio me resulto tedioso al haber trabajado poco con Git y archivos JSON pero conforme iba avanzando la práctica ya se iban aclarando las dudas. Elegimos Java por estar familiarizados con este lenguaje durante mas tiempo que otros y poder entender los pasos que debíamos seguir con mas facilidad.

Hemos aprendido a utilizar la herramienta GitHub ya que es muy útil para estar conectado con los demás miembros del grupo y poder subir diferentes versiones de la misma.

Para concluir las sesiones continuas de laboratorio han sido bastante útiles, ya que el profesor además de realizar un seguimiento de las practicas resolvía todas las dudas que se nos planteasen.

Manuel Morales Rodríguez

La realización de la práctica me ha resultado bastante amena ya que el tema del cubo de rubick me resulta bastante interesante.

Además, al tener la posibilidad de realizar la práctica en java, para mí personalmente ha sido más fácil que si la hubiéramos tenido que realizar en otro lenguaje de programación obligatoriamente, ya que llevo familiarizado con java desde primero de carrera, por lo tanto es el lenguaje que más domino.

Finalmente, comentar y agradecer que el tema de que la resolución de los árboles se de primero en clase teórica ya que ayuda muchísimo a la hora de hacer el programa