Ejercicio de prácticas de EDA del 2015.05.14

Rubik 2x2x2

Entrega

Entrega via **Campus Virtual** ó como un pull-request vía Github a https://github.com/manuel-freire/rubik con tu *nombre de grupo* en un comentario; envía tus soluciones antes de las 23:55h del lunes 18 de Mayo. Este ejercicio es **por parejas**.

Enunciado

Resuelve un cubo de Rubik. Escribe una clase **Rubik** que soporte las siguientes operaciones:

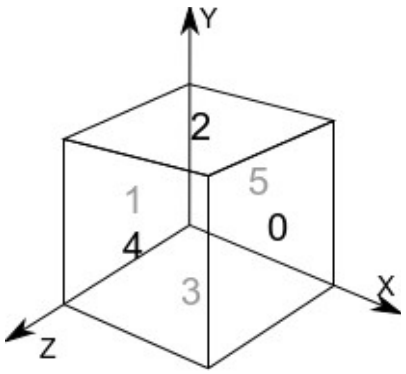

Fig 1: El orden de las caras a la hora de leer/escribir cubitos. En gris, las caras ocultas (1, 3, y 5).

- **Rubik**(2, *s*): constructor; inicializa un cubo de 2x2x2 a partir de la cadena *s*, que contendrá 8 (=2x2x2) bloques de caracteres, donde cada bloque describirá los colores de un cubito; así, un cubo resuelto podría ser ".d.e.f a..e.f .db..f a.b..f .d.ec. a..ec. .db.c. a.b.c." (los puntos indican "cara interna cuyo color da igual).

- **aCadena()**: devuelve una cadena que representa el estado del cubo. El formato es el mismo que el usado en el constructor.

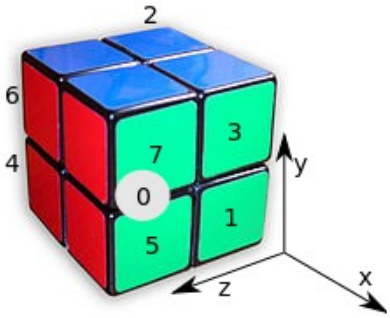

Fig 2: El orden de los cubitos a la hora de leer/escribir cubos enteros; el cubito 0 (oculto) es el más cercano al origen

- **gira**(*e1*, *e2*, *n*): donde *e1* y *e2* son ejes (ya sean 'x', 'y', ó 'z') y *n* es un nivel, que puede ser 0 ó 1. Gira una loncha del cubo (indicada por *n*: si *n*=0, será la loncha más cercana al origen de coordenadas la que gire; si *n*=1, será la loncha más alejada al orígen la que gire). A la hora de implementar *resuelve()*, los giros se representarían mediante cadenas del tipo "Gira y x 0".

- **resuelve**(*n*): devuelve una lista de las operaciones necesarias para restaurar el cubo actual a un estado resuelto. La lista contendrá una secuencia mínima (= no debe existir otra más corta) de cadenas que describen giros que, de aplicarse una tras otra y en orden, convertirían el cubo actual (que no debe ser modificado) en uno resuelto(). Debe devolver una lista vacía si ya está resuelto ó no es posible resolver el cubo en *n* ó menos operaciones. Recomendación: usa vuelta atrás ó búsqueda en anchura (= "por niveles") para este apartado.

- **resuelto**(): devuelve "true" si (y sólo si) cada cara visible tiene un solo color.

Calificación

Por la parte *pública* de la clase, 1 punto (+1 si usas "const" donde corresponde). Por una parte *privada* que permita implementar todas las operaciones, 3 puntos. Por una *implementación* correcta, 6 puntos (3 de ellos por "resuelve"). Y por dar la *complejidad* correcta en cada operación, 1 punto. En total, puedes sacar 12/10 en la parte básica.

Si consigues manejar 3x3x3, +4 puntos; y +2 más si soportas *dxdxd*. En estos casos, pasarás la dimensión *d* como primer argumento del constructor, y habrá *d* lonchas que se puedan girar en gira(); pero no debe cambiar nada más de la parte pública. Si ayudas a compañeros *enviándoles* "pull request" en github corrigiendo algún error (que no "solucionándole la práctica" ni "cambiándole un comentario"), hasta +2 puntos (*aceptar* estos "pull requests" no afecta nota). Con opcionales, puedes sacar hasta un 20/10.

Tu código debe compilar para ser calificado.