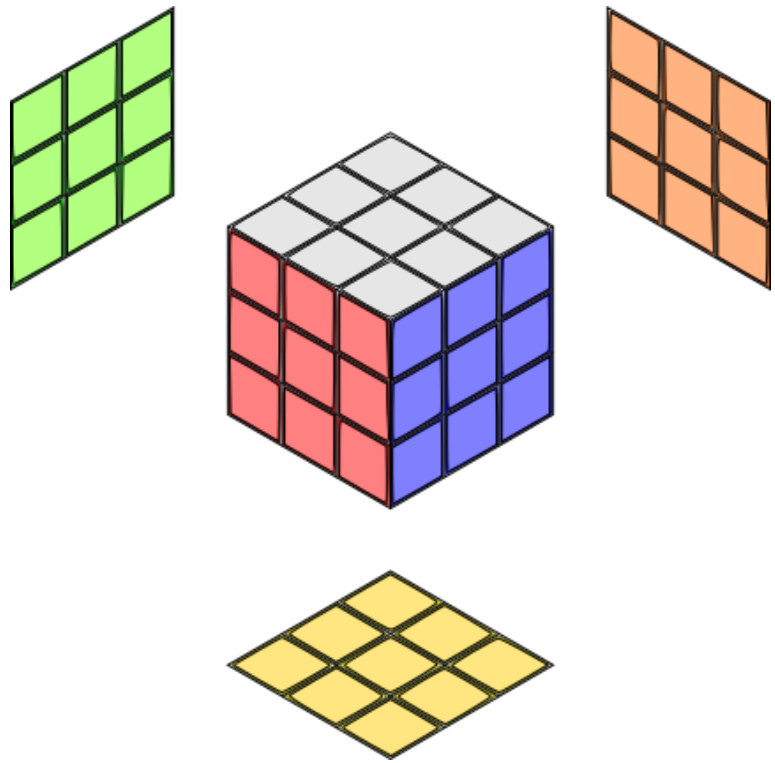
Il cubo di Rubik (e come risolverlo)

Stefano Angeleri, Alessandro Menti, Mattia Zago

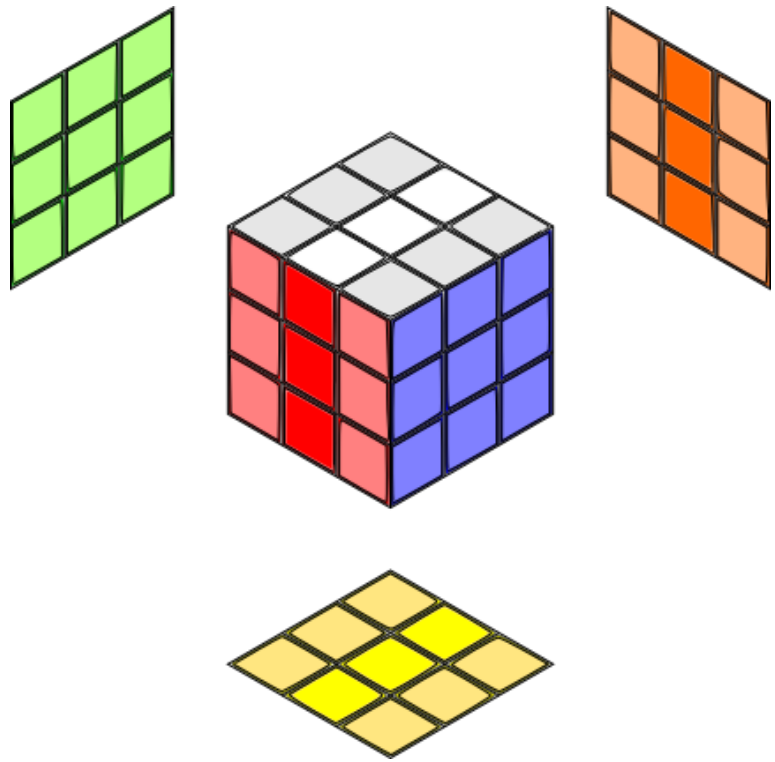
Alcune definizioni

- Considereremo un cubo 3×3
- Ogni faccia (*side*) ha un colore standard a essa associato (vedi figura)
- Ognuno dei nove pezzi di ogni faccia è detto *facelet*
- Il cubo ha 3 colonne/righe (*columns*/*rows*), 3 colonne laterali (*lateral columns*), 4 angoli (*corners*) e 8 spigoli (*edges*)



Alcune definizioni

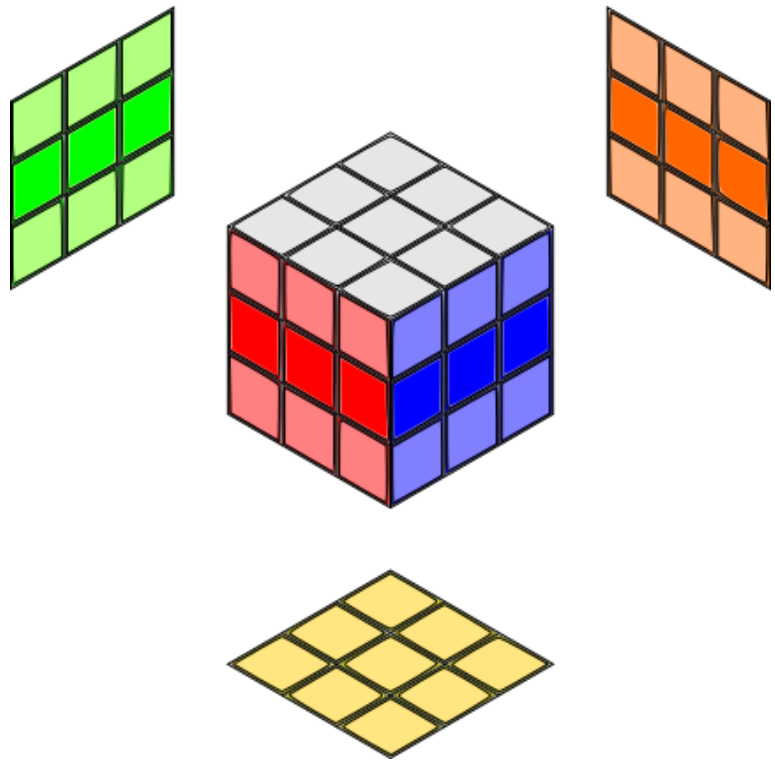
- Considereremo un cubo 3×3
- Ogni faccia (*side*) ha un colore standard a essa associato (vedi figura)
- Ognuno dei nove pezzi di ogni faccia è detto *facelet*
- Il cubo ha 3 colonne/righe (*columns*/*rows*), 3 colonne laterali (*lateral columns*), 4 angoli (*corners*) e 8 spigoli (*edges*)



Colonna

Alcune definizioni

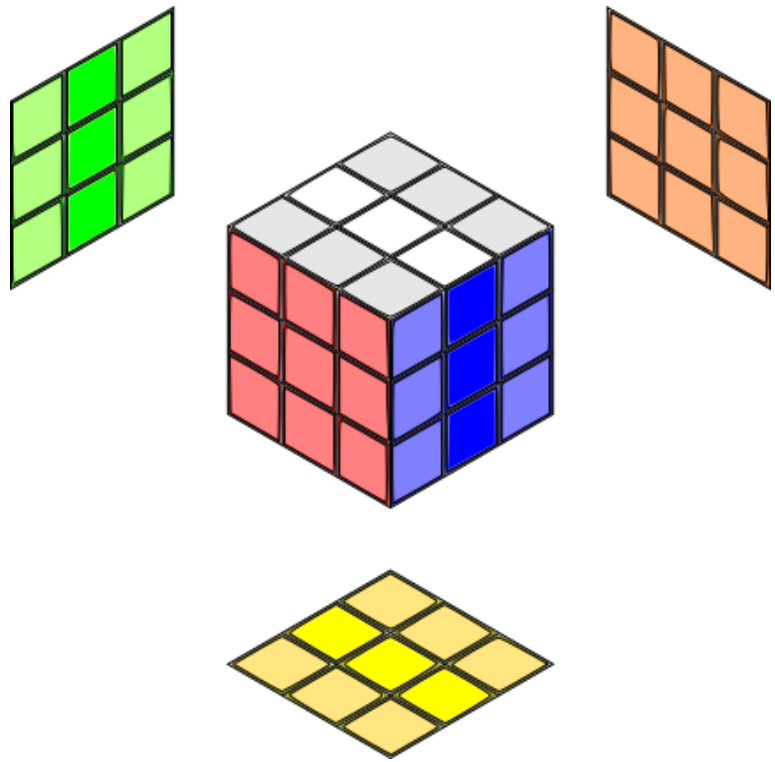
- Considereremo un cubo 3×3
- Ogni faccia (*side*) ha un colore standard a essa associato (vedi figura)
- Ognuno dei nove pezzi di ogni faccia è detto *facelet*
- Il cubo ha 3 colonne/righe (*columns*/*rows*), 3 colonne laterali (*lateral columns*), 4 angoli (*corners*) e 8 spigoli (*edges*)



Riga

Alcune definizioni

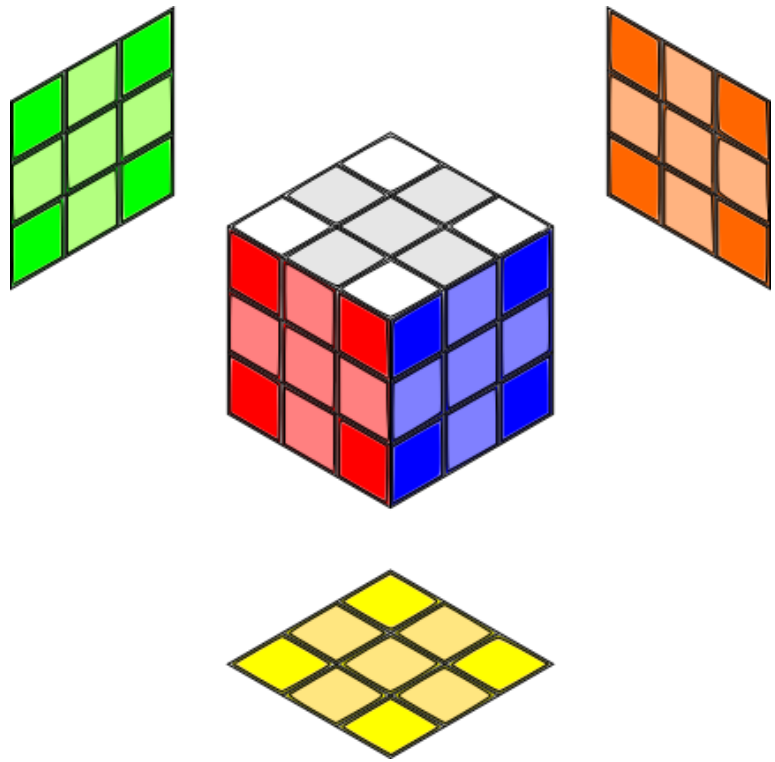
- Considereremo un cubo 3×3
- Ogni faccia (*side*) ha un colore standard a essa associato (vedi figura)
- Ognuno dei nove pezzi di ogni faccia è detto *facelet*
- Il cubo ha 3 colonne/righe (*columns*/*rows*), 3 colonne laterali (*lateral columns*), 4 angoli (*corners*) e 8 spigoli (*edges*)



Colonna laterale

Alcune definizioni

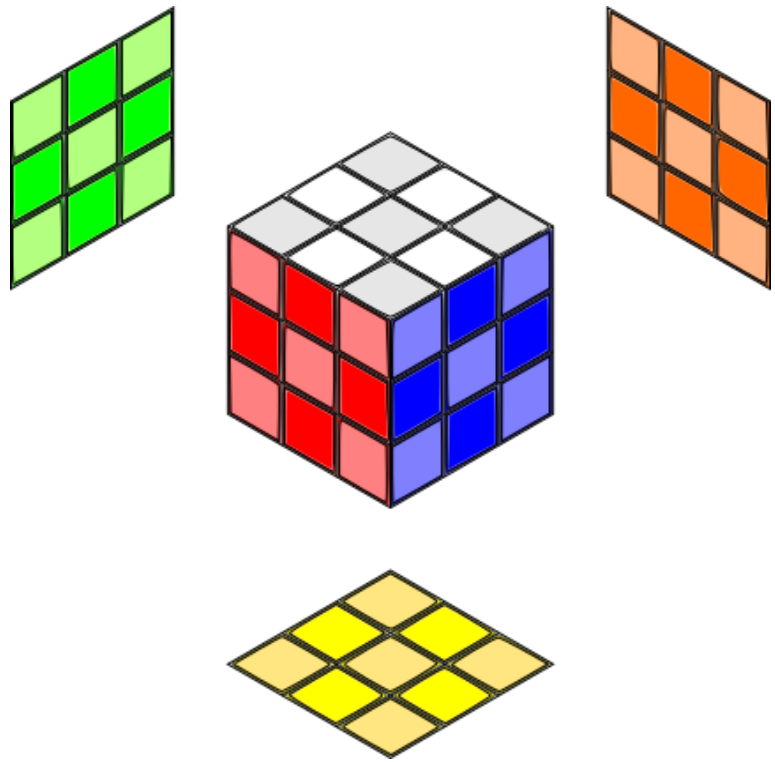
- Considereremo un cubo 3×3
- Ogni faccia (*side*) ha un colore standard a essa associato (vedi figura)
- Ognuno dei nove pezzi di ogni faccia è detto *facelet*
- Il cubo ha 3 colonne/righe (*columns*/*rows*), 3 colonne laterali (*lateral columns*), 4 angoli (*corners*) e 8 spigoli (*edges*)



Angolo

Alcune definizioni

- Considereremo un cubo 3×3
- Ogni faccia (*side*) ha un colore standard a essa associato (vedi figura)
- Ognuno dei nove pezzi di ogni faccia è detto *facelet*
- Il cubo ha 3 colonne/righe (*columns*/*rows*), 3 colonne laterali (*lateral columns*), 4 angoli (*corners*) e 8 spigoli (*edges*)



Spigolo

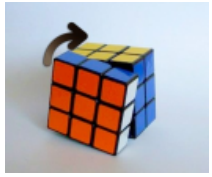
Il problema

Riarrangia il cubo (ruotando righe, colonne e/o colonne laterali) finché tutte le facelet su ogni faccia non hanno lo stesso colore.

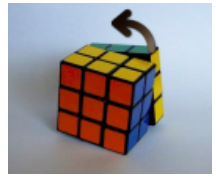
Notazione di Singmaster

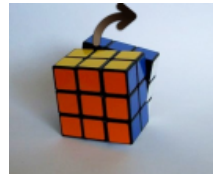
- Ogni faccia è descritta da una lettera: **F** (Front), **B** (Back), **U** (Up), **D** (Down), **L** (Left), **R** (Right)
- Ogni mossa può essere vista come una rotazione di un quarto di giro di una faccia in senso orario (N.B.: si assume che il solutore abbia la faccia di fronte a sé): **U** = ruota la faccia "Up" di un quarto di giro in senso orario
- Il simbolo $'$ indica una rotazione in senso antiorario
- Le rotazioni di righe/colonne/colonne laterali centrali sono denotate da **M** (*middle* — livello fra L e R), **E** (*equator* — livello fra U e D), **S** (*standing* — livello fra F e B)
- Per denotare le rotazioni del cubo si usano altre lettere: **X** (rotazione su R), **Y** (rotazione su U), **Z** (rotazione su F)

Notazione di Singmaster


F


F'


B


B'


U


U'


D


D'


L

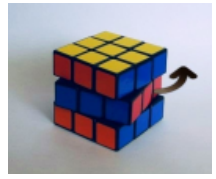

L'

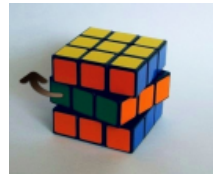

R


R'


M

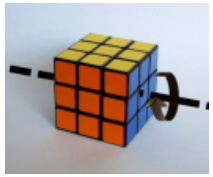

M'

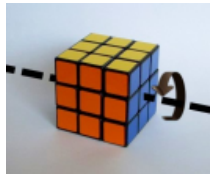

E

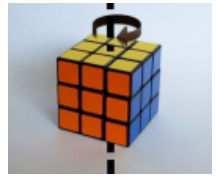

E'

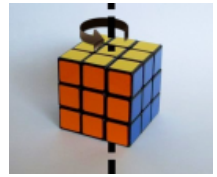

S

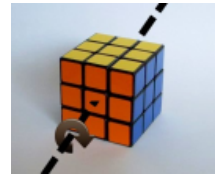

S'

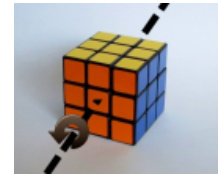

X


X'


Y


Y'


Z


Z'

Il nostro modello

- Il cubo è memorizzato in un oggetto `RubikCubeModel`
- Ogni faccia è memorizzata in un array 2×2; le righe/colonne sono numerate dall'alto verso il basso e da sinistra a destra (supponendo che il solutore abbia la faccia di fronte)
- `getSide` determina la faccia che in tale momento ha il colore dato
- `getFace` recupera il colore di una facelet
- Altri metodi autoesplicativi: `get3DEdge` (per gli angoli), `get3DEdgeFacelet` (facelet di un angolo), `getCorner`, `getCornerFacelet`
- Metodi `rotate*` per ruotare il cubo
- Test standard: `isInStandardConfiguration`, `isWithSaneColors`, `isSolved`, `isCornerInPlace`, `isCornerInPlaceMaybeFlipped`, `isEdgeInPlace`, `isEdgeInPlaceMaybeFlipped`

Mosse di Singmaster

- Sono state implementate le mosse standard di Singmaster
- Ogni mossa (per motivi di astrazione) è una sottoclasse di `Move`
- Il costruttore accetta come parametri il modello del cubo (in modo che il cubo originale rimanga inalterato) e un parametro `reversed` (per sapere se la mossa è diretta o inversa)
- Per applicare una mossa, basta crearla e chiamare `perform`/`reverse`:

```
(new B(m, reversed)).perform();
```

- Ogni mossa genera un evento per comunicare i cambiamenti all'interfaccia

Strategie di risoluzione

- Sono sottoclassi di `ResolutionStrategy`
- Accettano un cubo (`RubikCubeModel`) e restituiscono una lista di mosse da eseguire per risolvere il problema (`getNextMoves`)

Pathfinding

- Possiamo rappresentare i possibili svolgimenti di una partita con un grafo i cui nodi sono la configurazione del cubo in un dato momento; due nodi sono collegati se e solo se ci si può recare da una configurazione a un'altra con una sola mossa
- L'idea alla base della maggior parte degli algoritmi di risoluzione del cubo è quella di trovare un cammino su tale albero avente origine nella radice (configurazione iniziale) e termini nel cubo risolto

A*

- Mantengo due liste: una *(open list)* che contiene i nodi ancora da valutare, un'altra *(closed list)* per i nodi già valutati
- Fisso una funzione *costo* per ogni nodo: esso deve essere, intuitivamente, tanto minore quanto minore è il "disordine" rispetto al cubo risolto
- Calcolo per ogni nodo un indice

$$f(n) = g(n) + h(n)$$

  dove $g(n)$ è il costo minimo dei nodi nella closed list e $h(n)$ è una stima del costo del nodo n
- A ogni passo sposto il nodo considerato dalla open alla closed list (ad eccezione del caso in cui $g(n)$ diminuisca) e genero i suoi successori (tenendo traccia di tale legame)
- Al termine, estraendo il nodo con il minimo $f(n)$ e seguendo i genitori ho la sequenza di mosse cercata (al contrario)

IDA*

- A* ha un difetto: richiede di esplorare tutto l'albero
- Non fattibile per il cubo di Rubik (x configurazioni possibili!)
- Basta non analizzare i rami per cui non crediamo di ottenere risultati
- IDA* fa questo: per ogni nodo, se $f(n)$ è maggiore di un certo valore limite che fissiamo, pota il ramo

Fissare un'euristica

Rimane solo un problema: fissare un'euristica $h(n)$ sufficientemente buona per i nostri scopi

L'algoritmo di Thistletwaite

- Thistletwaite nel xxxx scoprì che era possibile dividere le mosse in quattro gruppi:

12

- Si noti che ogni gruppo è chiaramente incluso nel precedente e che l'ultimo gruppo comprende il cubo risolto
- Idea: portare il cubo da una configurazione risolubile con tutte le mosse possibili (primo gruppo) in una risolubile solamente con mosse appartenenti al secondo gruppo, quindi al terzo...
- Esaminando le configurazioni possibili si può ricavare un buon coefficiente euristico FIXME cosa pesa di più?

L'algoritmo a due fasi di Kociemba

L'algoritmo di Singmaster