IMPLEMENTASI ALGORITMA *LAYER-BY-LAYER* UNTUK MENYELESAIKAN RUBIK'S CUBE DALAM KODE PROGRAM

Khandar William

Program Studi Teknik Informatika
Sekolah Tinggi Teknik Elektro dan Informatika
Institut Teknologi Bandung
Alamat: Jalan Ganesha No. 10 Bandung
e-mail: if16022@students.if.itb.ac.id

ABSTRAK

Banyak orang menganggap bahwa permainan Rubik's Cube memerlukan IQ yang tinggi untuk menyelesaikannya, padahal sudah terdapat banyak algoritma yang ditemukan untuk menyelesaikan permainan ini, mulai dari yang paling mudah tetapi memakan banyak langkah, hingga yang paling rumit tetapi memakan langkah yang sedikit. Salah satu algoritma paling mudah untuk digunakan adalah algoritma *Layer-by-Layer*, algoritma ini merupakan salah satu varian dari algoritma *Greedy*.
Makalah ini membahas salah satu cara mengimplementasikan algoritma *layer-by-layer* dalam bentuk kode program (*source code*). Bahasa pemrograman yang digunakan dalam makalah ini adalah bahasa C++.
Program jadi dan kode sumbernya (*source code*) dapat diunduh dari pranala (*link*) yang terdapat di bagian Referensi makalah ini [7]. Makalah ini mengsumsikan bahwa pembaca sudah tidak asing dengan permainan Rubik's Cube.

Kata kunci: Rubik's cube, algoritma, *layer-by-layer*, *source code*, *greedy*, implementasi.

1. PENDAHULUAN

Rubik's Cube ditemukan oleh Erno Rubik dan telah dipatenkan atas namanya [5]. Benda ini sekilas terlihat sebagai sebuah kubus yang terdiri atas 27 kubus kecil, padahal sebenarnya hanya terdapat 26 kubus kecil karena kubus kecil yang paling dalam tidak pernah tersentuh. Cara memainkan kubus ini adalah dengan mengacak sisi-sisinya lalu mengembalikannya ke keadaan tersusun, di mana keenam sisinya memiliki warna yang *uniform*.

Hingga saat ini, banyak orang yang menganggap bahwa Rubik's Cube hanya dapat diselesaikan oleh orang-orang yang memiliki IQ tinggi, padahal sudah banyak algoritma yang ditemukan untuk menyelesaikan Rubik's Cube, sehingga siapa pun dapat menyelesaikannya. Salah satu algoritma yang termudah adalah algoritma *Layer-by-Layer* yang menyelesaikan Rubik's Cube mulai layer pertama, kedua, lalu ketiga. Selain itu, ada juga algoritma *Fridrich* yang memerlukan sekitar 120 algoritma untuk dihapal namun dapat menyelesaikan Rubik's Cube dalam rata-rata 50 gerakan.

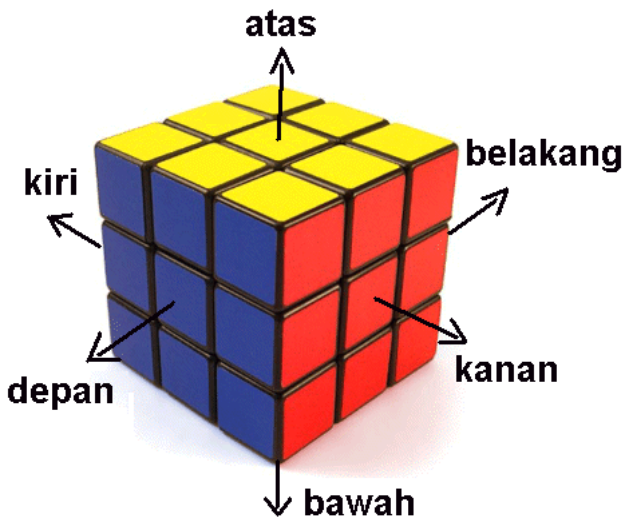
Karena solusi Rubik's Cube telah ditemukan dalam bentuk algoritma, maka program komputer untuk menyelesaikannya pun dapat dibuat. Makalah ini membahas bagaimana cara membuat program komputer yang mengimplementasikan algoritma *Layer-by-Layer* untuk menyelesaikan Rubik's Cube.

2. DASAR TEORI

2.1 Terminologi yang Dipakai

Terminologi yang biasa dipakai untuk menjelaskan suatu algoritma Rubik's Cube sebagai berikut:

- Rubik's Cube terdiri atas enam buah sisi: depan (*front*), belakang (*back*), kiri (*left*), kanan (*right*), atas (*up*), dan bawah (*down*).
- Gerakan memutar sisi Rubik's Cube searah jarum jam disimbolkan dengan huruf pertama dari nama sisi tersebut dalam bahasa Inggris. Untuk yang berlawanan arah dengan jarum jam, ditambahkan tanda petik ('). Contohnya, F berarti memutar sisi depan (*front*) searah jarum jam, sedangkan L' berarti memutar sisi kiri (*left*) berlawanan arah dengan jarum jam.
- Gerakan memutar seluruh kubus tidak memiliki notasi yang tetap, sehingga makalah ini menggunakan "*RotateRight*", "*RotateLeft*", "*RotateUp*", dan "*RotateDown*". Nama gerakan tersebut sudah menjelaskan maknanya.



Gambar 1. Penamaan sisi-sisi pada Rubik's Cube

2.2 Algoritma *Layer-by-Layer*

Secara garis besar, algoritma *Layer-by-Layer* dapat ditulis sebagai berikut:

a. Selesaikan *layer* pertama/atas.
   a. Bentuk *cross* di sisi atas, sesuaikan dengan warna keempat sisi di samping.
   b. Isi keempat sudut atas dengan kubus yang sesuai.
b. Selesaikan *layer* kedua/tengah.
   a. Isi keempat kubus pada *layer* kedua dengan kubus yang sesuai.
c. Selesaikan *layer* ketiga/bawah.
   a. Bentuk *cross* di sisi bawah, tanpa merusak kedua *layer* di atas.
   b. Tempatkan keempat sudut bawah di tempat sebenarnya, disesuaikan dengan warna dari ketiga sisi yang bersisian dengannya.
   c. Bentuk supaya sisi bawah memiliki warna yang *uniform.*
   d. Pertukarkan kubus-kubus yang belum sesuai pada tempatnya.

Untuk penjelasan lengkap dari tiap-tiap langkah algoritma ini, akan dijelaskan pada bab tiga disertai dengan potongan *source code* yang mengimplementasikannya.

3. IMPLEMENTASI

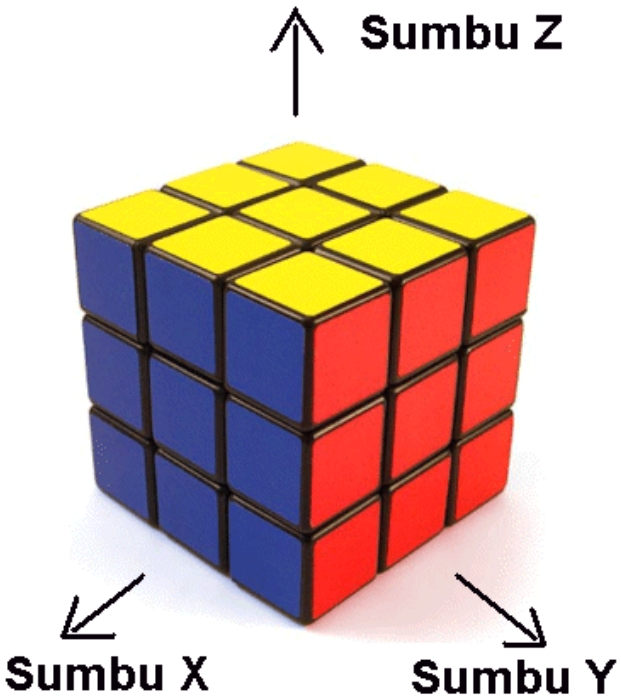
3.1 Representasi Rubik's Cube

Untuk dapat mengimplementasikan algoritma ini, harus dibuat dulu representasi dari Rubik's Cube dalam bentuk kode program juga. Makalah ini merepresentasikan Rubik's Cube sebagai sebuah larik (*array*) berdimensi empat yang berisi warna dari tiap sisi kubus kecil.

Potongan *source code* yang menggambarkan hal ini adalah:

```
typedef enum Color {GREEN='G', RED='R',
       YELLOW='Y', ORANGE='O', WHITE='W',
       BLUE='B', HIDDEN='H'};
class Rubik {...
... Color cubes[3][3][3][3]; ...
... };
```

Makna dari larik ini adalah keenam sisi yang dimiliki oleh tiap-tiap kubus kecil dikelompokkan menjadi tiga, yaitu sisi pada sumbu X, sumbu Y, dan sumbu Z.



Gambar 2. Penamaan sumbu pada Rubik's Cube

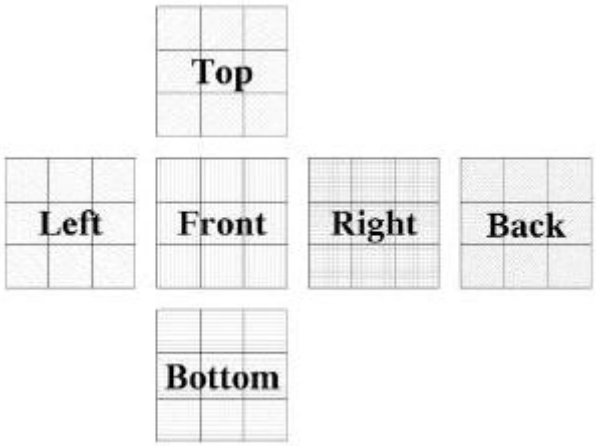
Meskipun sebuah kubus kecil memiliki enam buah sisi, jumlah sisi maksimum yang dapat terlihat hanya tiga, yaitu pada bagian sudut-sudut Rubik's Cube. Karena itu, digunakan representasi larik empat dimensi yang berisi warna di mana masing-masing dimensi memiliki tiga buah properti. Koordinat dari sebuah warna adalah sebagai berikut:

`cubes[i][j][k][l]`

i: 0 untuk depan, 1 untuk antara depan dan belakang, 2 untuk belakang
j: 0 untuk kiri, 1 untuk antara kiri dan kanan, 2 untuk kanan
k: 0 untuk atas, 1 untuk antara atas dan bawah, 2 untuk bawah
l: 0 untuk sumbu X, 1 untuk sumbu Y, 2 untuk sumbu Z

Warna ketujuh, yaitu HIDDEN digunakan untuk menandakan bahwa warna kubus kecil pada sumbu tersebut tidak terlihat. Sebagai contoh, pada gambar Rubik's Cube di atas, `cubes[0][1][1]` akan memiliki warna BLUE pada sumbu X, tetapi HIDDEN untuk sumbu Y dan sumbu Z.

Untuk menampilkan Rubik's Cube, digunakan tampilan dalam bentuk dua dimensi, yaitu:



Gambar 3. Tampilan dua dimensi Rubik's Cube

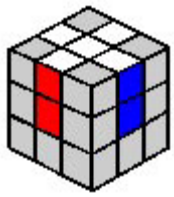
3.2 Implementasi Algoritma *Layer-by-Layer*

Implementasi dari algoritma *Layer-by-Layer* secara garis besar dapat dilihat pada potongan *source code* berikut:

```
void Rubik::solveCube() {
       solveFirst();
       solveSecond();
       solveThird();
}

void Rubik::solveFirst() {
       makeCrossTop();
       fillCornerTop();
}

void Rubik::solveSecond() {
       fillEdgeMiddle();
}

void Rubik::solveThird() {
       makeCrossBottom();
       reorderCornerBottom();
       makeFullBottom();
       reorderEdgeBottom();
}
```

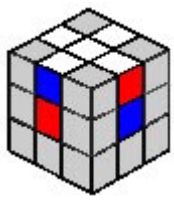
3.2.1 *Layer* Pertama

Langkah pertama dalam menyelesaikan *layer* pertama/atas adalah membentuk sebuah *cross* di sisi atas, seperti pada gambar ini.



Gambar 4. *Cross* yang sukses dibentuk

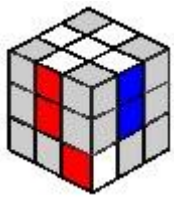
Langkah ini cukup mudah untuk dilakukan sendiri sehingga tidak perlu dijelaskan secara mendalam. Hanya saja, satu hal yang harus diperhatikan, yaitu warna di samping keempat buah ujung *cross* tersebut harus serupa dengan warna di bawahnya, berikut adalah contoh *cross* yang salah.



Gambar 5. *Cross* yang gagal dibentuk

Implementasi dari langkah ini dapat dilihat pada *source code* disertakan pada lampiran, baris 860-865 dan 596-680.

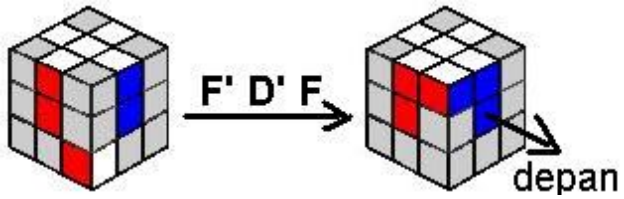
Setelah *cross* berhasil dibentuk, langkah selanjutnya adalah mengisi sudut-sudut bagian atas dengan kubus yang sesuai tanpa merusak *cross* tersebut. Untuk bagian ini, pertama-tama tempatkan kubus yang seharusnya menjadi sudut tersebut di bagian bawahnya dan putar Rubik's Cube hingga kubus tersebut berada di bawah kiri depan, contoh:



Gambar 6. Tempatkan sudut di bawah tempat sebenarnya

Lalu lakukan algoritma berikut berdasarkan letak sisi yang warnanya sama dengan warna *cross*:

- jika menghadap ke depan, F' D' F
- jika menghadap ke kiri, L D L'
- jika menghadap ke bawah, F' D D F D F' D' F



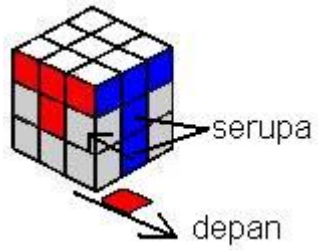
Gambar 7. Contoh salah satu kasus

Lakukan langkah ini hingga keempat sudut tersebut benar pada tempatnya dan selesaikan layer pertama. Implementasi dari langkah ini dapat dilihat pada *source code*, pada baris 881-886 dan 682-715.

3.2.2 *Layer* Kedua

Tidak banyak yang perlu dilakukan di sini, karena hanya tedapat empat buah kubus yang perlu dibenarkan tempatnya, bahkan hanya terdapat dua algoritma yang perlu dihapal.

Untuk setiap kubus yang tidak benar letaknya (pada layer kedua), tempatkan kubus yang seharusnya berada pada tempat tersebut di bawah salah satu sisi Rubik's Cube di mana warna sisi kubus tersebut serupa dengan warna sisi kubus di atasnya. Lalu, putar Rubik's Cube hingga sisi tersebut menjadi sisi depan. Sebagai contoh:



Gambar 8. Letakkan di bawah salah satu sisi

Lakukan algoritma berikut berdasarkan di mana seharusnya kubus bawah tersebut diletakkan:

- jika diletakkan di kiri, F D F D F D' F' D' F'
- jika diletakkan di kanan, F' D' F' D' F' D F D F

Gambar 9. Contoh salah satu kasus

Lakukan langkah ini hingga keempat kubus pada *layer* kedua berada pada tempat sebenarnya. Seandainya ada kubus yang terbalik atau salah tempat pada *layer* kedua, tempatkan kubus tersebut pada *layer* ketiga dengan cara menukarnya dengan salah satu kubus di *layer* ketiga, setelah itu lakukan algoritma di atas seperti biasa, sebagai contoh:

Gambar 10. Ada kubus yang terbalik

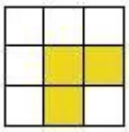
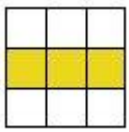
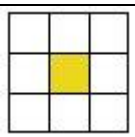
Implementasi dari langkah ini dapat dilihat pada *source code*, baris 888-893 dan 717-766.

3.2.3 *Layer* Ketiga

Layer ketiga adalah *layer* yang tersusah karena *layer* ini harus diselesaikan tanpa merusak kedua *layer* di atasnya. Untuk itu, cukup banyak algoritma yang harus dihapal untuk menyelesaikan *layer* terakhir ini.

Langkah pertama mirip dengan langkah pertama untuk *layer* pertama, yaitu membentuk *cross* pada sisi bawah. Jika pada sisi bawah belum terbentuk sebuah *cross*, ada tiga kondisi yang mungkin terjadi, yaitu:

Tabel 1 Kondisi dari *cross* pada sisi bawah dan solusi

Kondisi	Gambar
dua kubus bersebelahan belum membentuk *cross* Solusi: putar sisi bawah hingga berbentuk seperti gambar, lalu lakukan F D L D' L' F'	
dua kubus berseberangan belum membentuk *cross* Solusi: putar sisi bawah hingga berbentuk seperti gambar, lalu lakukan F L D L' D' F'	
empat kubus belum membentuk *cross* Solusi: lakukan salah satu algoritma di atas, setelah itu, salah satu kondisi di atas akan terpenuhi	

Implementasi dari langkah ini dapat dilihat pada *source code*, pada baris 867-879.

Setelah *cross* terbentuk, letakkan keempat sudut pada *layer* bawah di tempat yang seharusnya, tetapi tidak perlu menghadap arah yang sebenarnya. Sebagai contoh, sudut merah-putih-biru harus berada di antara sisi merah, putih, dan biru, tetapi ketiga sisi sudut ini tidak perlu menghadap arah yang benar.

Cara untuk membenarkan letak keempat sudut ini adalah dengan menukarkan letak dua buah sudut bawah. Hanya diperlukan satu buah algoritma untuk hal ini, algoritma ini menukar letak sudut depan-kiri-bawah

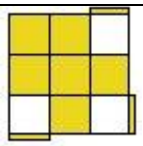
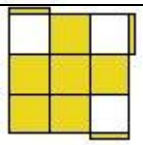
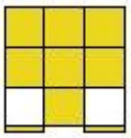
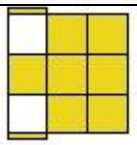
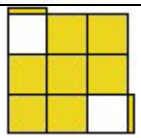
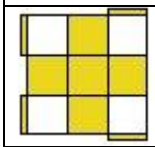
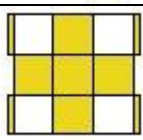
dengan sudut belakang-kiri-bawah tanpa merusak posisi layer pertama dan kedua. Algoritma tersebut adalah

R D' L' D R' D' L D D

Implementasi dari langkah ini dapat dilihat pada *source code*, pada baris 895-905.

Setelah keempat sudut berada pada posisi sebenarnya, langkah selanjutnya adalah membuat sisi bawah berwarna serupa. Untuk mempermudah, putar Rubik's Cube ke atas dua kali sehingga sisi bawah kini menjadi sisi atas. Jika sisi atas (tadinya sisi bawah) Rubik's Cube belum berwarna serupa, ada tujuh kasus yang mungkin terjadi, seperti dapat dilihat pada tabel di bawah ini. Jika tidak ada kondisi yang cocok, putar Rubik's Cube ke kiri atau ke kanan hingga sisi atasnya cocok dengan salah satu kondisi di bawah ini.

Tabel 2 Kondisi dari sisi atas dan solusi

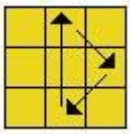
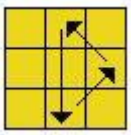
Kondisi	Solusi
	R' U' R U' R' U U R U U Ini adalah kondisi 1, setelah algoritma ini dilakukan, sisi atas akan langsung serupa
	R U R' U R U U R' U U Ini adalah kondisi 2, setelah algoritma ini dilakukan, sisi atas akan langsung serupa
	R U R' U R U U R' U U Setelah itu sisi atas akan menjadi kondisi 1 atau 2
	R U R' U R U U R' U U Setelah itu sisi atas akan menjadi kondisi 1 atau 2
	R' U' R U' R' U U R U U Setelah itu sisi atas akan menjadi kondisi 1 atau 2
	R U R' U R U U R' U U Setelah itu sisi atas akan menjadi kondisi 1 atau 2
	R U R' U R U U R' U U Setelah itu sisi atas akan menjadi kondisi 1 atau 2

Implementasi dari langkah ini dapat dilihat pada *source code*, baris 907-940.

Langkah terakhir dalam menyelesaikan *layer* ketiga (yang kini berada di atas) adalah menukar posisi kubus yang masih tidak berada pada tempatnya. Jika masih ada kubus yang salah tempat, ada dua kondisi yang mungkin, yaitu tiga kubus salah tempat atau empat kubus salah tempat.

Jika ada tiga kubus salah tempat, ada dua kondisi yang mungkin terjadi, seperti digambarkan pada tabel di bawah. Jika sisi atas tidak cocok dengan kedua gambar ini, putar Rubik's Cube ke kiri atau kanan hingga ada gambar yang cocok.

Tabel 3 Kondisi dari sisi atas dengan tiga kubus salah tempat dan solusi

Kondisi	Solusi
	R R U F B' R R F' B U R R
	R R U' F B' R R F' B U' R R

Jika ada empat kubus yang salah tempat, lakukan salah satu dari algoritma di atas (untuk tiga kubus) dan setelah itu kubus yang salah tempat akan menjadi tiga sehingga algoritma di atas dapat digunakan.

Implementasi dari langkah terakhir ini dapat dilihat pada *source code*, pada baris 942-962.

4. KESIMPULAN

Tidak benar anggapan yang mengatakan bahwa diperlukan IQ tinggi untuk menyelesaikan Rubik's Cube. Dengan ditemukannya algoritma seperti *Layer-by-Layer*, *Fridrich*, dan *Lars Petrus*, siapa pun dapat menyelesaikan Rubik's Cube. Bahkan, program untuk menyelesaikannya pun sudah banyak beredar di internet.

Program yang dibuat di dalam makalah ini dapat dikategorikan sebagai program yang sangat sederhana karena menggunakan algoritma yang paling sederhana juga, ditambah lagi antarmukanya yang masih *text-based*. Untuk pengembangan selanjutnya, program ini dapat diberikan antarmuka yang GUI dan tiga dimensi.

Source code yang mengimplementasikan algoritma *Layer-by-Layer* ini sudah terlampir pada makalah ini, tetapi tanpa program utama yang menjalankan antarmukanya. *Source code* yang lengkap beserta antarmuka dapat diunduh pada pranala yang terdapat pada bagian Referensi [7].

REFERENSI

[1] Munir, Rinaldi, "Diktat Kuliah IF2251 Strategi Algoritmik", ITB Press, 2007.

[2] Weigang, Jim, "*Implementing Rubik's Cube: An Exercise in Hybrid Programming*", *Computer Language, January issue*, 1986.

[3] http://peter.stillhq.com/jasmine/rubikscubesolution.html (tanggal akses: 16 Mei 2008)

[4] http://www.ryanheise.com/cube/beginner.html (tanggal akses: 16 Mei 2008)

[5] http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=4378116 (tanggal akses: 19 Mei 2008)

[6] http://en.wikipedia.org/wiki/Rubik%27s_Cube (tanggal akses: 19 Mei 2008)

[7] http://students.if.itb.ac.id/~if16022/programrubik.zip (*source code* lengkap program jadinya)