# Penerapan Algoritma Branch and Bound pada Permainan Pocket Cube

Alif Ijlal Wafi/13515122
Program Studi Sarjana Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515122@std.stei.itb.ac.id

***Abstract*—Makalah ini membahas tentang peran penggunaan algoritma Branch and Bound untuk optimisasi penyelesaian permasalahan salah satu *puzzle* kombinatoral yaitu *Pocket Cube*. Pada permasalahan ini, algoritma Branch and Bound dipakai untuk menentukan cara *solving Pocket Cube* dengan langkah paling minimum dengan pendekatan yang ditentukan oleh penulis.**

***Keywords—branch and boundt; pocket cube; gerakan minimum; 3D manhattan distance***

## I. PENDAHULUAN

*Puzzle* kombinatorial merupakan salah satu jenis permainan yang cukup popular di kalangan masyarakat. Beberapa contoh dari *puzzle* kombinatorial antara lain 4x4 15-*puzzle* dan Klotski. *Puzzle-puzzle* ini memiliki kesamaan yakni terdiri atas bagian-bagian yang dapat ditukar tempatnya satu sama lain dengan kombinasi berbagai macam gerakan. Gerakan-gerakan yang dimaksud tidak sembarangan, melainkan terikat dalam kondisi tertentu, biasanya berupa akibat dari konstruksi fisik masing-masing *puzzle*. Kondisi menang keduanya pun mirip, yaitu mengembalikan bagian-bagian tersebut kembali ke posisi masing-masing yang seharusnya dari posisi acak namun harus dapat dicapai dengan gerakan-gerakan yang ditentukan sebelumnya. Kedua ciri-ciri tersebut adalah hal-hal yang dapat mendefinisikan apa itu *puzzle* kombinatorial

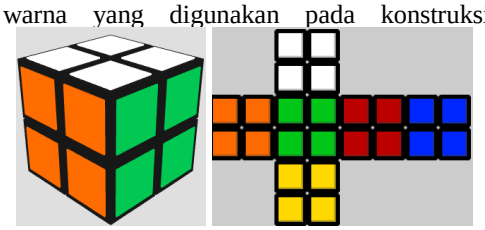
Salah satu *puzzle* kombinatorial yang umum dikenal adalah *Rubik's cube,* atau cukup disebut rubik di Indonesia. Konstruksi rubik berupa sebuah kubus yang terbagi atas 26 bagian kubus sama besar, dengan 9 bagian di tiap sisinya, menghasilkan 54 bagian yang harus ditata secara keseluruhan. Sejak diciptakannya rubik oleh Ernő Rubik pada tahun 1974, Rubik telah menjelma menjadi salah satu mainan dengan penjualan tertinggi hingga saat ini. Rubik juga menginspirasi pembuatan banyak variannya, salah satu diantaranya yaitu *Pocket Cube*, varian lebih sederhana dari rubik yang berdimensi 2x2x2 bagian dibandingkan dengan dimensi 3x3x3 rubik.

Dampak lain dari kepopuleran rubik dan variannya ialah adanya keinginan untuk menyelesaikan *puzzle* tersebut secepat-cepatnya, seperti halnya dengan jenis-jenis *puzzle* lain. Untuk mencapai tujuan tersebut, pemain-pemain rubik dan variannya tidak hanya membutuhkan kemampuan untuk menggerakkan rubik secara cepat, namun juga perlu dapat menemukan jumlah gerakan paling sedikit untuk mencapai kondisi *solved*. Salah satu pendekatan optimisasi yang dapat dilakukan adalah dengan menggunakan algoritma *Branch and Bound*. Pendekatan optimisasi ini tentunya akan berguna khususnya bagi pemain rubik berbasis komputer untuk dapat menentukan solusi dengan langkah minimum.

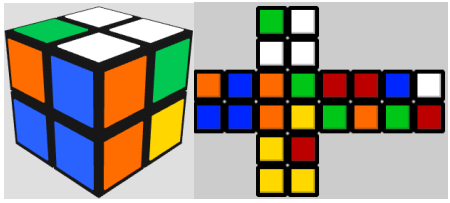
## II. DASAR TEORI

### A. *Gerakan-Gerakan yang diperbolehkan dan Kondisi Solve pada Pocket Cube*

.Struktur *Pocket Cube* sebagai varian 2x2x2 dari *Rubik's Cube* adalah kubus yang terdiri atas 8 bagian sama besar, memiliki 6 sisi dan terbagi atas 4 bagian di setiap sisinya. Kondisi *solve* pada *Pocket Cube*, seperti halnya *Rubik's Cube* adalah setiap bagian dari sebuah sisi memiliki warna yang sama, dengan warna masing-masing sisi terikat pada skema warna yang digunakan pada konstruksi *Pocket Cube*.



(a)



(b)

Gambar 2.1 (a) Kondisi *solve Pocket Cube* (b) Salah satu contoh kondisi acak *Pocket Cube*

(Sumber : https://ruwix.com/online-puzzle-simulators/2x2x2-pocket-cube-simulator.php , https://www.grubiks.com/puzzles/rubiks/mini-cube-2x2x2/ )

Pada *Pocket Cube*, notasi gerakan yang umum digunakan adalah :

- F (*Front clockwise*)
- F' (*Front counter-clockwise*)
- R (*Right clockwise*)
- R' (*Rightt counter-clockwise*)
- U (*Up clockwise*)
- U' (*Up counter-clockwise*)
- B (*Back clockwise*)
- B' (*Back counter-clockwise*)
- L (*Left clockwise*)
- L' (*Left counter-clockwise*)
- D (*Down clockwise*)
- D' (*Down counter-clockwise*)

Pada skema yang digunakan di contoh sebelumnya, pembagian sisinya pada kondisi *solve* adalah:

- *Front* : Hijau
- *Right* : Merah
- *Up* : Putih
- *Back* : Biru
- *Left* : Oranye
- *Down* : Kuning

Pertukaran sisi walaupun secara natural digunakan oleh pemain manusia untuk kepentingan observasi, tidak dilakukan oleh pemain berbasis komputer. Pada pembahasan selanjutnya skema di atas akan dijadikan acuan kondisi *solve*.

*B. Representasi Pocket Cube pada Komputer*

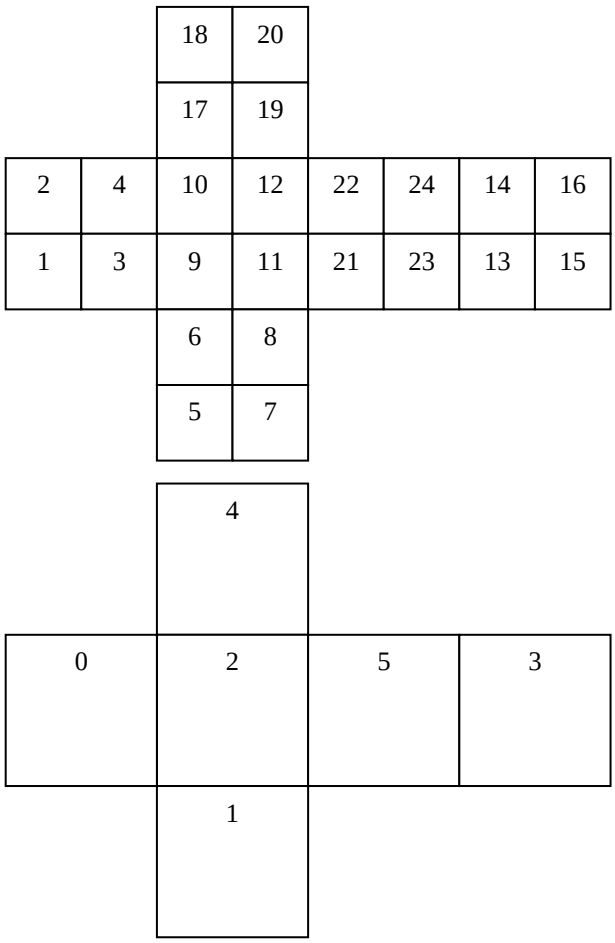
Representasi *Pocket Cube* pada program komputer dapat bermacam-macam bentuknya. Sebagai referensi, untuk *Rubik's Cube* ada setidaknya tujuh alternatif cara : *array* 3 dimensi 3x3x3 yang terdiri atas 3 digit bilangan bulat, *array* 3 dimensi 6x3x3 yang terdiri atas *literals*, *matrix* 5x12 yang terdiri atas *literals*, *II-by-II sparse literal matrix*, *vector* dengan 54 elemen, *array* 4 dimensi, dan *nested array* 3x3x3.

Representasi *Pocket Cube* yang diajukan penulis adalah dengan objek *Cube* dan *Face*, dengan pseudocode :

```
class Face
    - matrix : int[2][2]
    - side : int
    +set(x : int, y : int, val : int) : nothing
    +get(x : int, y : int)
    +rotate(clockwise : boolean) : nothing

class Cube
    -faces : Face[6]
    +isEqual (c : Cube) : boolean
    +manhattanDist() : int
    + turn(face : int, clockwise : boolean)
```

Representasi yang diajukan diatas menggunakan *array of matrix*, dengan menggunakan bilangan bulat sebagai representasi suatu warna pada posisi dan keterikatan tertentu.




Gambar 2.2. Representasi *Pocket Cube* yang diajukan penulis pada kondisi *solve*

Dengan representasi yang demikian, setiap angka mewakili sebuah bagian warna dengan properti yang unik dengan bagian warna lain. Misalnya, angka 2 dan angka 4 sama-sama mewakili bagian berwarna oranye, namun properti masing-masing berbeda : angka 2 mewakili bagian berwarna oranye yang bersisian dengan warna putih dan biru, sedangkan angka 4 mewakili bagian berwarna oranye yang bersisian dengan warna putih dan hijau.

Tatanan *Face* yang demikian dimaksudkan untuk mempermudah pencarian apakah suatu bagian bertolak belakang dengan bagian lain atau tidak, dengan aturan apabila jumlah indeks kedua sisi adalah 5 maka kedua sisi tersebut seharusnya bersisian. Dengan adanya kesesuaian angka

representasi warna dengan *Face* masing-masing, maka hubungan tersebut dapat dicek dengan lebih cepat.

### C. *Algoritma Branch and Bound*

Algoritma *Branch and Bound* merupakan salah satu paradigma desain algoritma untuk menemukan solusi optimal dalam ruang solusi yang berfokus pada permasalahan optimisasi. Prinsipnya dari algoritma *Branch and Bound* adalah membangkitkan pohon solusi dinamis dari suatu permasalahan (*Branch)* dengan tidak melanggar batasan persoalan (*Bound)* untuk mengoptimasi suatu fungsi objektif.

Algoritma *Branch and Bound* secara umum adalah :

1. Masukkan simpul akar dalam antrian simpul Q. Jika simpul akar = simpul solusi, solusi ditemukan. Berhenti.
2. Jika antrian Q kosong, tidak ada solusi. Berhenti.
3. Jika antrian Q idak kosong, pilih simpul i dari Q dengan cost(i) terkecil dengan keterurutan yang sudah ditentukan apabila terdapat beberapa i yang memenuhi.
4. Jika simpul i adalah simpul solusi, solusi ditemukan., berhenti.
5. Jika simpul i bukan simpul solusi, bangkitkan semua anak-anaknya. Jika simpul i tidak mempunyai anak, kembali ke 2.
6. Untuk semua simpul j anak dari simpul i, masukkan j beserta cost(j) ke dalam Q. Kembali ke 2.

Dapat dilihat bahwa algoritma Branch and Bound membutuhkan suatu struktur data yang wajib digunakan yaitu priority queue yang diurutkan berdasarkan cost tergantung dari permasalahan yang akan dioptimasi. Untuk permasalahan pada Pocket Cube, priority queue jelas diurutkan dari yang terkecil.

Bagian penting lainnya dalam konstruksi algoritma *Branch and Bound* adalah penentuan fungsi cost setiap simpul, yang biasa dituliskan secara matematis sebagai :

$$cost(i) = f(i) + heuristic(i)$$

*cost(i)* = ongkos untuk simpul i

*f(i)* = ongkos untuk mencapai simpul i dari akar

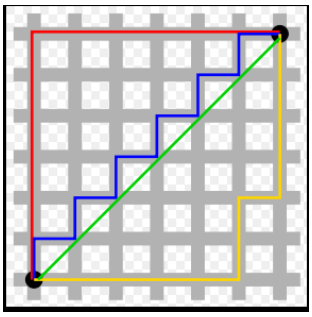
*heuristic(i)* = taksiran ongkos mencapai simpul tujuan dari simpul i.

Pada permasalahan *Pocket Cube* ini, f(i) didefinisikan sebagai jumlah gerakan dari posisi acak awal hingga simpul i, sedangkan heuristic(i) didefinisikan sebagai suatu nilai yang berupa 3D Manhattan distance.

### D. *Manhattan Distance*

*Manhattan Distance* merupakan salah satu fungsi geometri yang didefinisikan sebagai jarak antar dua titik berdasarkan jumlah selisih dari masing masing sumbu, atau secara matemaris :

$$f((x1,y1),(x2,y2)) = |x1 - x2| + |y1-y2|$$



Gambar 2.3. Contoh representasi jarak 2 titik
Sumber : https://en.wikipedia.org/wiki/Taxicab_geometry#/media/File:Manhattan_distance.svg

*Manhattan Distance* dipakai untuk merepresentasikan jumlah langkah sesungguhnya untuk mencapai suatu titik tertentu dari suatu titik tertentu yang lain. Contoh lebih jelasnya dapat dilihat pada gambar di samping. *Euqledian Distance* yang direpresentasikan dengan garis berwarna hijau memperlihatkan jarak real 8.49 , namun jarak yang harus ditempuh sesungguhnya adalah *Manhattan Distance* yaitu 12, yang direpresentasikan dengan rute berwarna merah, biru dan kuning.

Varian *Manhattan Distance* untuk bidang 3 dimensi serupa dengan fungsi awalnya, yaitu :

$$f((x1,y1,z1),(x2,y2,z2)) = |x1 - x2| + |y1-y2| + |z1-z2|$$

Namun, varian 3D *Manhattan Distance* ini tentu tidak optimal untuk digunakan pada permasalahan *Pocket Cube* akibat adanya kemungkinan permutasi 3 sisi warna pada satu bagian. Oleh karena itu, definisi *Manhattan Distance* pada setuap bagian *Pocket Cube* yang diajukan penulis diubah menjadi jumlah langkah yang harus ditempuh untuk memposisikannya pada posisi bagian yang lain, dirumuskan menjadi :

$$f((x1,y1,s1),(x2,y2,s2)) = g(s1,s2) + |x1 - x2| + |y1-y2|$$

Dengan g(s1,s2) bernilai 0 apabila terdapat pada satu sisi, 1 apabila kedua sisi yang dimaksud bersisian, dan bernilai 2 apabila kedua sisi yang dimaksud bertolak belakang. Dengan hubungan yang sudah dijelaskan pada bagian B, dapat dituliskan :

$$g(s1, s2) = \begin{cases} 0, s1 - s2 = 0 \\ 2, s1 + s2 = 5 \\ 1, otherwise \end{cases}$$

Kemudian, nilai total dari keseluruhan *Manhattan Distance* dibagi 4 akibat setiap gerakan menggerakkan 4 bagian.

## III. STUDI KASUS: APLIKASI ALGORITMA *BRANCH AND BOUND* PADA SUATU MASALAH *SCRAMBLED POCKET CUBE*

Algoritma *Branch and Bound* dapat digunakan untuk menghasilkan solusi untuk menyelesaian permasalahan pada *Pocket Cube* khususnya untuk pemain komputer. Selanjutnya pada bagian ini akan dibahas bagaimana suatu permasalahan pada *Pocket Cube* dapat diselesaikan dengan komputer menggunakan algoritma *Branch and Bound*.

### A. *Ekstraksi Informasi*

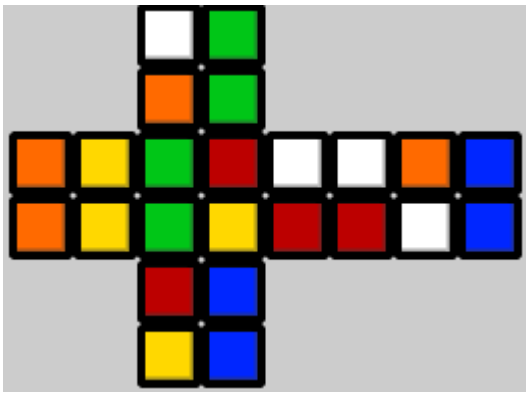
Sebelum melanjutkan ke tahap menemukan solusi, komputer tentu diberikan informasi terlebih dahulu mengenai *state* masalah saat ini dan bagaimana menyimpan informasi tersebut dalam struktur data yang sebelumnya diajukan.

Pertama, komputer harus mendapatkan informasi korespondensi antara angka yang merepresentasikan suatu bagian dengan bagian yang bersangkutan. Hal ini dapat dilakukan baik dengan cara sederhana yaitu memberi informasi secara langsung pada perangkat lunak maupun dengan cara membuat pemain komputer tersebut mengobservasi terlebih dahulu *Pocket Cube* yang sudah dalam kondisi *solve*. Informasi yang perlu didapat kurang lebih adalah sebagai berikut :

Tabel 3.1 Informasi Korespondensi Representasi Angka

| Angka | Warna | Properti |
|---|---|---|
| 1 | Oranye | Bersisian dengan kuning, biru |
| 2 | Oranye | Bersisian dengan biru, putih |
| 3 | Oranye | Bersisian dengan hijau, kuning |
| 4 | Oranye | Bersisian dengan hijau, putih |
| 5 | Kuning | Bersisian dengan oranye, biru |
| 6 | Kuning | Bersisian dengan hijau, oranye |
| 7 | Kuning | Bersisian dengan merah, biru |
| 8 | Kuning | Bersisian dengan merah, hijau |
| 9 | Hijau | Bersisian dengan oranye, kuning |
| 10 | Hijau | Bersisian dengan putih, oranye |
| 11 | Hijau | Bersisian dengan kuning, merah |
| 12 | Hijau | Bersisian dengan merah, putih |
| 13 | Biru | Bersisian dengan merah, kuning |
| 14 | Biru | Bersisian dengan putih, merah |
| 15 | Biru | Bersisian dengan kuning, oranye |
| 16 | Biru | Bersisian dengan putih, oranye |
| 17 | Putih | Bersisian dengan oranye, hijau |
| 18 | Putih | Bersisian dengan biru, oranye |
| 19 | Putih | Bersisian dengan hijau, merah |
| 20 | Putih | Bersisian dengan merah, biru |
| 21 | Merah | Bersisian dengan hijau, kuning |
| 22 | Merah | Bersisian dengan putih, hijau |
| 23 | Merah | Bersisian dengan kuning, biru |
| 24 | Merah | Bersisian dengan biru, putih |

Informasi di atas selanjutnya disimpan agar tidak perlu dicari lagi pada percobaan berikutnya. Kemudian, hal berikutnya yang dilakukan adalah ekstraksi masalah, yaitu *state Pocket Cube* yang dalam kondisi acak dan menyimpan informasinya ke struktur data yang telah diajukan berdasarkan informasi korespondensi. Contoh kasus yang akan dibahas misalnya sebagai berikut :



Gambar 3.1 Contoh kasus permasalahan

(Sumber : https://ruwix.com/online-puzzle-simulators/2x2x2-pocket-cube-simulator.php )

Maka representasi pada struktur data adalah sebagai berikut:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | 18 | 10 | | | | |
| | | 3 | 12 | | | | |
| 2 | 6 | 9 | 22 | 19 | 17 | 4 | 16 |
| 1 | 8 | 11 | 7 | 23 | 24 | 20 | 15 |
| | | 21 | 13 | | | | |
| | | 5 | 14 | | | | |

Dengan total *Manhattan Distance* sebesar 30

*B. Processing dengan Branch and Bound*

Setelah didapatkan data informasi seperti di atas, tahap selanjutnya yaitu membangkitkan pohon solusi secara bertahap menggunakan Algoritma *Branch and Bound* dengan definisi cost tiap simpul yaitu :

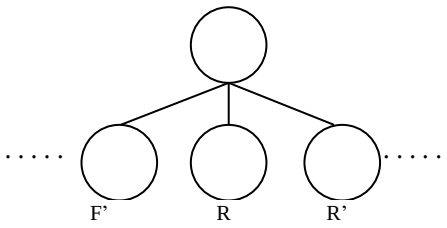
$$cost(i) = f(i) + g(i)$$

*f(i)* = jumlah gerakan hingga i

*g(i)* = (jumlah total *Manhattan Distance/4)*

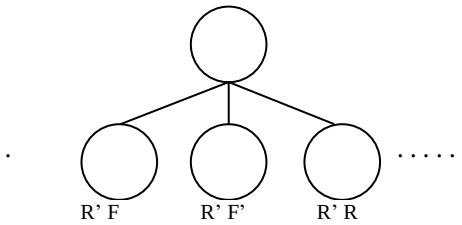
Tahapan :

1. Hidupkan seluruh simpul dan hitung *bound*, berurutan dari F, F' dan seterusnya hingga D'




(Tahapan disimplifikasi)

2. Simpul R' memiliki *cost* terkecil, namun bukan simpul solusi. ekspansi R'




(Tahapan disimplifikasi)

3. Simpul R' F' memiliki *cost* terkecil dan ternyata merupakan simpul solusi. Akhiri pencarian.

## IV. PEMBAHASAN

Algoritma *Branch and Bound* dapat digunakan unruk menyelesaika permasalahan kombinatorial seperti *Pocket Cube*, karena algoritma tersebut akan dapat selalu menghasilkan solusi optimum dalam waktu yang dapat ditolerir karena pembangkitan pohon solusi yang berdasarkan *cost* serta dilakukannya pemangkasan simpul-simpul yang tidak mengarah ke solusi optimal setelah ditemukannya simpul solusi. Hal ini tentu berbeda apabila kita menggunakan algoritma sederhana yang lain dengan *cost* pencarian tinggi, misalnya *brute-force*, BFS dan DFS. Penggunaan *brute-force* dan DFS tentunya tidak optimal pada permasalahan kombinatorial karena besarnya kemungkinan untuk terjadi adanya pencarian tidak terhingga akibat langkah-langkah yang bisa diulang terus-menerus. Sedangkan BFS dalam permasalahan kombinatorial yang sederhana mungkin dapat dijadikan alternatif selain *Branch and Bound*, namun pembangkitan simpul yang berdasarkan urutan gerakan awal saja mengakibatkan *cost* waktu pencarian yang cukup tinggi dan sangat dipengaruhi oleh kasus permasalahan jika dibandingkan dengan *Branch and Bound*.

Namun, bukan berarti algoritma berbasis *Branch and Bound* tidak memiliki kelemahan dalam penyelesaian permasalahan kombinatorial. Penentuan fungsi heuristik dalam algoritma *Branch and Bound* adalah suatu hal yang sangat penting dan dapat berakibat fatal apabila tidak dirumuskan secara benar. Misalnya dalam kasus persoalan *Pocket Cube* ini, fungsi heuristik *Manhattan Distance* total perlu dibagi 4, bukan digunakan langsung. Apabila tidak dibagi 4, maka dalam pembangkitan simpul-simpul untuk mencari simpul solusi algoritma tersebut praktis berlaku sebagaimana BFS. Hal ini diakibatkan nilai heuristik yang sangat besar bila dibandingkan dengan ongkos langkah yang dilakukan untuk mencapai tahapan tersebut. Kesalahan penentuan fungsi heuristik dapat disebabkan oleh berbagai hal. Dalam kasus ini, kesalahan penentuan tersebut terjadi umumnya karena penentu fungsi hanya melihat jumlah langkah yang dibutuhkan tiap bagian untuk mencapai tujuan aslinya. Pada praktiknya, setiap langkah menggerakkan 4 bagian sekaligus, sehingga jumlah langkah yang ditentukan tadi sebenarnya hanya membutuhkan seperempatnya.

## V. KESIMPULAN

Pada bagian-bagian di atas telah dijelaskan mengenai bagaimana algoritma *Branch and Bound* dapat digunakan untuk menyelesaikan permasalahan *Pocket Cube*. Algoritma tersebut akan selalu dapat menghasilkan solusi karena ruang solusi yang dibangkitkan terbatas serta makin diperkecil secara bertahap akibat adanya *bound* sehingga optimisasi yang dilakukan optimal.

## UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya sehingga penulis dapat menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih kepada kedua orangtua dan keluarga yang mendukung penulisan makalah ini sebagai salah satu kewajiban kuliah. Selain itu, penulis juga menyampaikan terima kasih kepada para dosen pengampu mata kuliah Strategi Algoritma, Dr. Nur Ulfa Maulidevi S.T., M.Sc., Dr. Masayu Leylia Khodra M.T., dan Dr. Ir. Rinaldi Munir, M.T. atas ilmu-ilmu yang telah disampaikan sehingga makalah ini dapat terselesaikan. Terakhir, terima kasih juga penulis sampaikan kepada pihak-pihak lain yang belum disebutkan yang membantu penulis dalam membuat makalah ini, baik penulis sadari maupun tidak.

## REFERENSI

[1] Munir, Rinaldi. 2004. Diktat IF2211 Strategi Algoritmik

[2] http://stackoverflow.com/questions/19455829/trying-to-solve-a-rubiks-cube-in-java, terakhir diakses pada 18 Mei 22.55

[3] https://heuristicswiki.wikispaces.com/Rubik%27s+cube, terakhir diakses pada 18 Mei 22.55

[4] https://heuristicswiki.wikispaces.com/3D+Manhattan+distance, terakhir diakses pada 18 Mei 22.55

[5] http://www.cs.princeton.edu/courses/archive/fall06/cos402/papers/korfrubik.pdf, terakhir diakses pada 18 Mei 22.55

[6] https://cs.stackexchange.com/questions/55660/heuristic-for-rubiks-cube, terakhir diakses pada 18 Mei 22.55

[7] https://www.speedsolving.com/wiki/index.php/Rubik%27s_Cube_Fact_sheet, terakhir diakses pada 18 Mei 22.55

[8] https://www.speedsolving.com/wiki/index.php/2x2x2 http://anttila.ca/michael/devilsalgorithm/, terakhir diakses pada 18 Mei 22.55

[9] http://stackoverflow.com/questions/500221/how-would-you-represent-a-rubiks-cube-in-code, terakhir diakses pada 18 Mei 22.55

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Mei 2017

Alif Ijlal Wafi
13515122