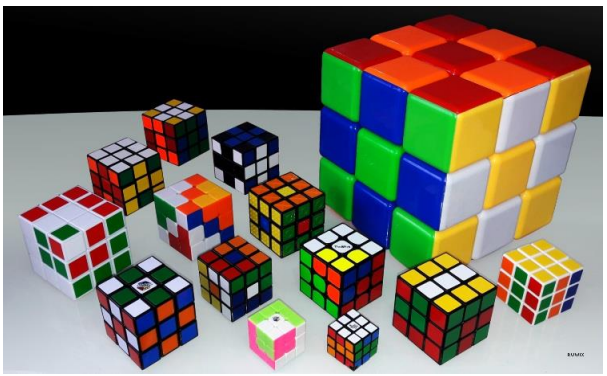
Perbandingan Penyelesaian *Rubik's Cube* Menggunakan Algoritma BFS dan *Branch and Bound*

Vijjasena / 13517084
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 bandung 40132, Indonesia

***Abstrak*—*Rubik's Cube* merupakan sebuah permainan puzzle yang memiliki banyak bentuk dan ukruan. Rubik's cube dtemukan pada tahun 1974 oleh seorang professor arsitektur dan juuga seorang pemahat dari Hungaria bernama Ernő Rubik. Pada umumnya *rubik's cube* berukuran 3x3 kubus tetati terdaat varian yang lebih kecil sehingga memudahkan untuk dibawa di dalam saku. Pada makalah ini akan dijabarkan perbandingan algoritma "*branch and bound*" dan algortima "*Breadth First Search*" dalam menyelesaikan rubik's cube.**

Kata Kunci*—*Branch and bound, Breadth First Search, BFS, Rubik's cube, algoritma

I. PENDAHULUAN



Gambar 1.1 Macam-macam Rubik[1]

Rubik's Cube diciptakan pada 1974 oleh professor Ernő Rubik karena ingin menolong murid-muridnya untuk memaham lebih dalam mengenai ruang tiga dimensi[2]. Pada awalnya Rubik's hanya berukuran 3x3 tetapi mucul model-model baru yang memiliki ukuran yang berbeda seperti rubik's 2x2, 4x4, 6x6, ada juga yang memiliki bentuk bentuk yang aneh dan bukan kubus seperti pada asalnya.

Gambar 1.2 Rubik unik yang bukan berbentuk kubus[3]

Saat ini rubiks memiliki turnamen internasional yang diikuti dari berbagai macam negara. Apa yang diperlombakan dalam turnamen rubiks ini? Tentu saja kecepatan seseorang dalam menyelesaikan rubiks yang telah disediakan. Rekor tercepat dipegang oleh seoran pria asal Melbourne dengan nama Feliks Zemdegs. Ia dapat menyelesaikan rubik dengan waktu penyelesaian selama 4,73 detik dan waktu tersebut juga merupaka rekor dunia[4].

Metode memainkan Rubiks ini juga bermacam-macam. Jika orang biasa memainkannya dengan mata terbuka dan memperhatikan setiap kotak warna yang ada pada kubik, ada teknik *blindfold* yang memainkan rubik dengan mata tertutup namun pemain tetap melihat keadaan rubik saat telah selesai diacak dari keadaan asalnya.

Selain metode untuk memainkan ada juga metode untuk menyelesaikan rubik ini. Metode yang dikembangkan juga tidak hanya satu, tetapi ada beberapa metode yang telah dikembangkan dengan keterangan yang cukup jelas dalam pengeksekusiannya. Salah satu contoh metode yang cukup terkenal adalah metode Fridrich. Metode ini menjelaskan setiap stepnya dengan baik dan cukup mudah unutk dimengerti.

Permainan dengan umur yang dapat dibilang cukup tua ini tentu saja memiliki aplikasi yang dapat menyelesaikan rubik tersebut dalam waktu yang cepat, bahkan tidak sampai hitungan menit untuk menyelesaikan rubik standart. Di dalam dunia informatika terdapat beberapa algoritma yang dapat menyelesaikan rubik ini dengan mengubah setiap keadaan dari rubik dianggap sebagai sebuah state yang dapat disusun menjadi sebuah graf yang pada salah satu daun mengandung bentuk awal dari rubik sebelum rubik

tersebut diacak. Algoritma yang dimaksud adalah Breath First Search(BFS), Depth First Search, Bactracking, dan juga Branch and Bound.

Algoritma-algoritma tersebut merupakan algoritma yang dapat dikatakan memiliki performa lebih baik dibandingkan dengan menggunakan algoritma brute force yang mencoba setiap kemungkinan yang mungkin dijalankan oleh algoritma brute force.

II. Dasar Teori

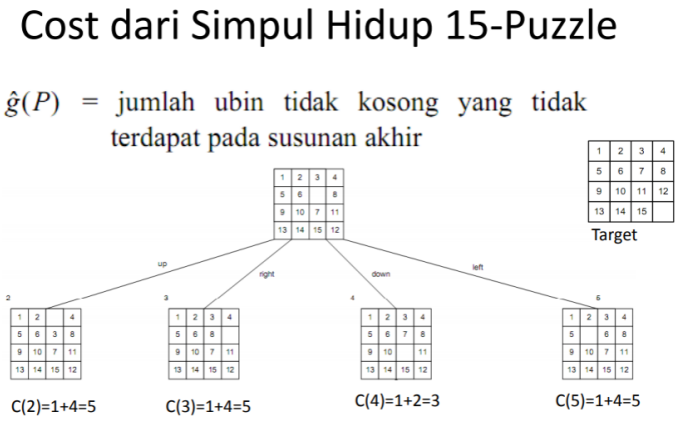
A. *Branch and Bound*

Branch and bound merupakan sebuah algoritma pencarian yang menggunakan *cost* sebaga bahan pertimbangan jalur mana yang akan ditelusuri.*cost* memiliki rumus sebagai berikut.

C(i) = f(i) + g(i)

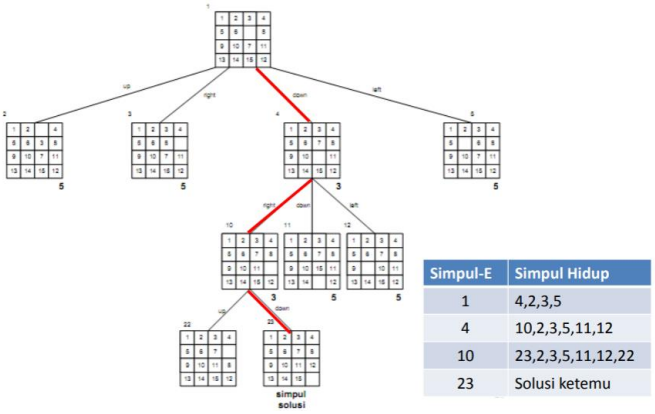
Dengan C(i) adalah *cost* untuk simpul i, f(i) merupakan ongkos dari simpul akar untuk menuju simpul i, dan dan g(i) meupakan ongkos untuk mencapai simpul tujuan dari simpul i yang merupakan taksiran lintasan terpendek menuju simpul tujuan.

Karena pada algoritma pencarian branch and bound digunakan *cost*, maka setelah membangkitkan 'anak' dari simpul akar, pembangkitan selanjutnya didasarkan oleh kepemilikan cost yang paling kecil. Oleh karena itu branch and bound lebih optimal dari BFS. Berikut contoh penghitungan *cost* pada pohon percarian di persoalan 15-puzzle.



Gambar 2.1 Contoh kalkulasi *cost* pada 15-puzzle[5]

Pada gambar dijelsakan bahwa g(i) merupakan jumlah kotak yang tidak berada pada tempat yang semestinya. Untuk itu g(2) untuk simpul 2 bernilai 4 karena ada 4 buah ubin yang tidak terdapat pada posisi seperti pada di target. Untuk f(2) bernilai 1 karena langkah yang dibutuhkan untuk sampai dari akar(simpul 1) ke simpul 2 hanyalah 1 langkah. Dengan demikian, *cost* yang dimiliki oleh simpul 2 adalah C(2) = f(2)+g(2) = 4 + 1 = 5. Berikut ini lengkanya dalam pencarian branch and bound.



Simpul-E	Simpul Hidup
1	4,2,3,5
4	10,2,3,5,11,12
10	23,2,3,5,11,12,22
23	Solusi ketemu

Gambar 2.2 Gambar pencarian *Branch and bound* 15-puzzle[5]

Pada algoritma *Branch and bound*, apabila telah ditemukan maka *cost* dari simpul solusi yang pertama ditemukan menjadi acuan apakah suatu simpul harus diberhentikan penurunannya atau masih boleh untuk diturunkan anaknya. Oleh karena itu, bila simpul solusi yang pertama kali ditemukan bernilai 25 maka simpul-simpul yang boleh diturunkan anaknya hanya yang memiliki *cost* kurang dari atau sama dengan 25.

Dalam pengimplementasiannya algoritma *branch and bound* menggunakan struktur data *priority queue* karena dalam pemmilihan simpul mana yang diturunkan berdasarkan *cost* terkecil. Dalam *priority queue* dapat di-representasika *cost* sebagai prioritas dari suatu simpul yang mana semakin mendekati 1, maka simpul tersebut akan berada semakn didepan pada antrian tersebut

Karena pada makalah ini algortima branch and bound akan digunakan untuk menyelesaikan rubik, berikut adalah langkah langkah yang akan dilakukan.

1. Simpul akar akan dimasukan ke dalam *priority queue* dan jika simpul tersebut merupakan solusi yang dicari maka pencrian akan diberhentikan.
2. Jika pada antrian kosong dan tidak ada solusi maka pencarian akan diberhentikan
3. Jika pada antrian tidak kosong maka akan diambil head dari antrian karena memiliki *cost* paling kecil.
4. Jika simpul I merupakan simpul solusi maka pencarian akan berhenti, jika bukan maka akan dibangkitkan anak-anak dari simpul terssebut dan bola simpul I tidak memiliki anak yang dapat diturunkan(node daun) maka akan kembali ke langkag 2.
5. Untik setiap simpul anak yang diturunkan pada langkah 4, dicari *cost* dari masing-masing anak dan masukan ke dalam *priority queue*.
6. Kembali ke langkah 2.

B. Breadth First Search

Algoritma lain yang dapat digunakan untuk menyelesaikan puzzle rubik dengan cara yang cukup serupa dengan *Branch and Bound* adalah *Breadth First Search* atau sering disebut dengan BFS. Dalah hal ini, yang dimaksud dengan serupa adalah cara penyelesaian masalah menggunakan pohon ruang status dalam pencariannya.

Selain BFS terdapat *Depth First Search*(DFS) yang mungkin juga dapat digunakan untuk menyelesaikan puzzle rubik ini. Dalam algoritma BFS tidak digunakan perhitungan *cost* seperti pada *Branch and Bound*. Namun algoritma ini mengunjungi seluruh simpul yang bertetangga dengan simpul yang telah dikunjungi.

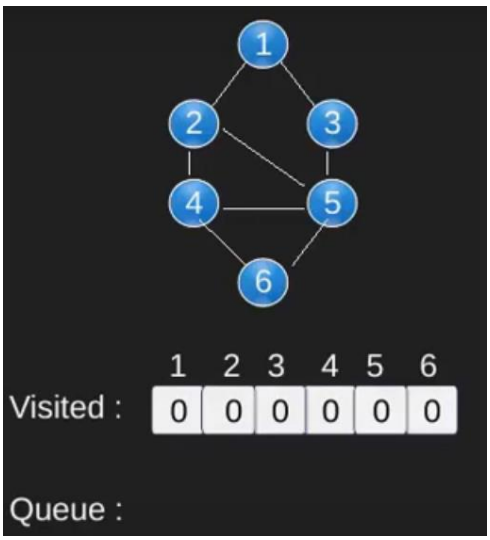
Dalam penngimplementasiannya digunakan struktur data queue untuk menyimpan urutan simpul yang akan dikunjungin nantinya oleh algoritma BFS. Selain menggunakan queue, digunakan juga sebuah array yang menyimpan simpul simpul yang telah dikunjungi sebelumnya sehingga pencarian tidak berakhir pada sebuah loop yang tidak terbatas. Alternatif lain dari menggunakan array adalah menggunkan matriks ketetanggaan.

Algoritma BFS dapat dieksekusi dengan mengikuti langkah-langkah sebagai berikut.

1. Kunjungi simpul v dan masukan ke dalam antrian. Pada table simpul yang telah dikunjungi masukan simpul v.
2. Masukan semua simpul yang belum dikunjungin serta bertetangga dengan simpul v ke dalam antrian.
3. Kunjungi simpul tetangga dari v sesuai degna uturan yang terdapat pada queue.
4. Kembali ke langkah 1.

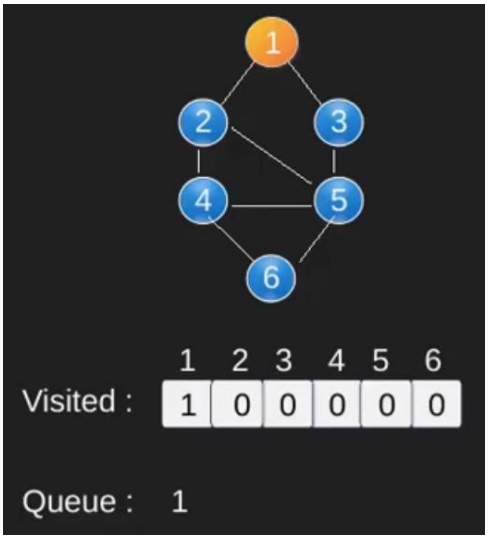
Bila dalam pencarian diinginkan untuk menyimpan rute dari simpul simpul yang mengarah ke simpul solusi dapat ditambahkan asal dari simpul yang dikunjungi dan masukan seluruh pencarian ke dalam sebuah array ataupun list. Untuk mencari rute dapat ditelusuri secara terbalik dari simpul tujuan ke simpul asal menggunakan informasi tambahan yang dimasukan ke dalam simpul. Untuk mencapai hal ini dapa digunkana struktur tubple ataupun membuat objek dan metode-metode yang ditambhan sendiri sehingga dapat memudahkan dalam pengolahan. Namun algoritma BFS ini kurang baik dalam hal kompleksitas ruang, karena bernilai $O(b^d)$ sendangkan DFS lebih baik dalam hal kompleksitas ruang yang bernilai O(bm).

Algoritma BFS ini pasti akan menghasilkan solusi yang paling optimal. Optimal yang dimaksud adalah jalur terpendek yang ada pada suatu graf dari simpul asal ke simpul tujuan. Meskipun dalam algoritma akan dikunjungi seluruh simpul yang ada dalam mencari simpul yang dicari. Berikut contoh penyelesaian sebuah graf dalam mencari simpul 6 dan dimulai dari simpul 1.



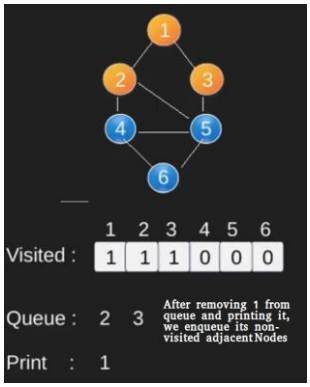
Gambar 2.3 Kondisi awal pencarian BFS[6]

Pada gambar diatas dapat dilihat telah disiapkan sebuah tabel untuk menyimpan simpul yang telah dikunjungi dan juga antrian.



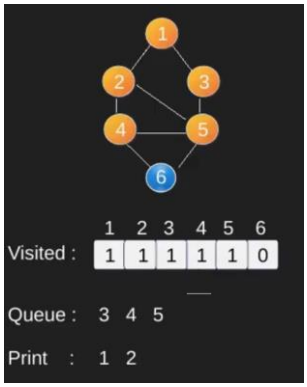
Gambar 2.4 Kondisi pencarian BFS langkah 2[6]

Pada gambar diatas, simpul awal pencarian dimasukan ke dalam aray yang telah dikunjungi dan juga dimasukan ke dalam antrian. Pada langkah selanjutnya akan dicetak simpul 1 dan diambil head dari antrian dan memasukan simpul yang bertetangga dengan simpul 1 yaitu simpul 2 dan simpul 3 ke dalam antrian.



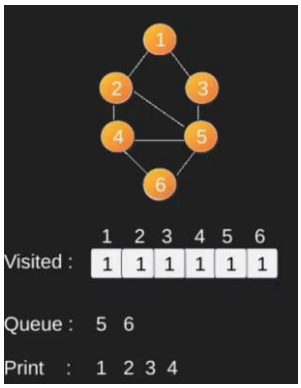
Gambar 2.5 Pencarian BFS Langkah 3 [6]

Setelah dimasukan simpul 2 dan simpul 3 ke dalam antrian dan juga ke dalam array yang telah dikunjungi maka, akan dikunjungi simpil-simpul tetangga dari simpul yang berada di head antrian dan masukan simpul-simpul tetangga tersebut ke dalam antrian dan juga array yang telah dikunjungi.



Gambar 2.6 Pencarian BFS Langkah 4[6]

Pada langkah selanjutnya adalah melakukan pencarian simpul tetangga dari simpul 3(head dari antrian). Namun karena simpul yang bertetangga dengan simpul 3 telah dikunjungi maka simpul 3 akan dicetak dan dilanjutkan oleh simpul 4.



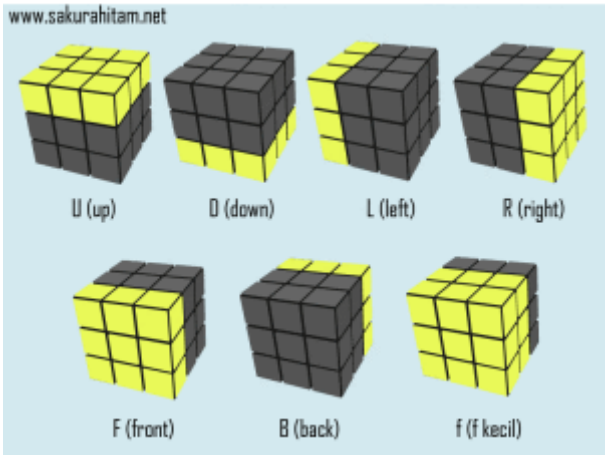
Gambar 2.7 Pencarian BFS Langkah 5[6]

Dapat dilihat pada array, seluruh simpul yang ada telah dikunjungi maka akan dicetak seluruh simpul yang tersisa di dalam antrian.

C. Metode Dalam Penyelesaian Rubik's Cube

Dalam penyelesaian rubik terdapat banyak metode, diataranya adalah Fridrich Method yang tebagi ke dalam dua bagian yaitu Advanced dan Beginner yang biasa disebut dengan layer by layer method, metode kedua adalah roux method, ketiga adalah ZZ method. Dalam langkah langkah menyelesaikan suatu ubik terdapat beberapa istilah yang harus diketaui yaitu arah perputaran dan symbol yang digunakan. Berikut ini symbol symbol yang digunakan.

- U : Bagian atas rubik diputar searah jarum jam
- D : Bagian bawah rubik diputar searah jarum jam
- R : Bagian kanan rubik diptar searah jarum jam
- L : Bagian kiri rubik diputar searah jarm jam
- F : Bagian depan rubik diputar searah jarum jam
- B : Bagian belakang rubik diputar searah jarum jam
- U' :Bagian atas diputar berlawanan jarum jam
- D' : Bagian bawah diputar berlawanan jarum jam
- R' : Bagian kanan diputar berlawanan jarum jam
- L' : Bagina kiri diputar berlawanan jarum jam
- F' : Bagian depan diputar berlawanan jarum jam
- B' : Bagian belakang diputar berlawanan jarum jam



Gambar 2.8 Bagian dari Rubik[7]

Selama langkah-langkah penyelesaian sebaiknya rubik tidak diubah prespektif melihatnya. Hal ini akan mengakibatkan langkah penyelesaian yang terdapat pada metode tidak dapat digunakan karena tampak dari rubik telah berubah.

Istilah langkah yang telah dijelaskan diatas akan diubah menjadi sebuah metode dari sebuah objek rubik sehinga dalam enyelesaian akan lebih mudah untuk dioperasikan.

III. Analisis Dan Hasil Data

A. *Algoritma*

Pada makalah ini akan dibandingkan antara algoritma Branch and bound dan BFS. Berikut penjabaran untuk algoritma branch and bound.

C(i) = f(i) + g(i). dengan g(i) merupakan jumlah petak yang tidak berada pada posisi yang seharusnya dan f(i) adalah jumlah langkah dari akar ke simpul tersebut. Berikut contohnya.

Gambar 3.1 Rubik Acak[8]

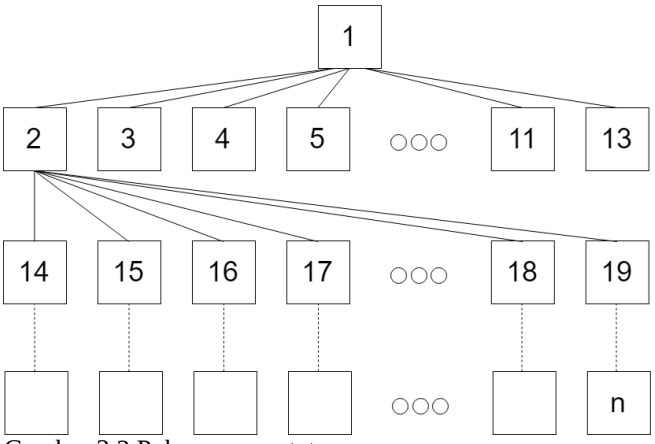
Pada gambar diatas nilai g(i) adalah 44 untuk mengetahui warna dari sebuah kotak dapat dilihat dari petak tengah dari sebuah kotak. Contoh, untuk kotak paling atas merupakan kotak yang seharusnya berwarna biru pasa semua petak yang ada di dalam kotak.

Untuk menyelesaikan persoalan rubik ini, berikut algoritma dari *Branch and bound* yang digunakan. Simpul akar pada permasalahan ini adalah rubik yang telah diacak terlebih dahulu. Dan simpul solusi merupakan keadaan dimara rubik telah tertata ulang.

1. Simpul akar akan dimasukan ke dalam *priority queue* dan jika simpul tersebut merupakan solusi yang dicari maka pencrian akan diberhentikan.
2. Jika pada antrian kosong dan tidak ada solusi maka pencarian akan diberhentikan
3. Jika pada antrian tidak kosong maka akan diambil *head* dari antrian karena memiliki *cost* paling kecil.
4. Jika simpul i merupakan simpul solusi maka pencarian akan berhenti, jika bukan maka akan dibangkitkan anak-anak dari simpul tersebut dan dihitung *cost* dari simpul simpul anak dengan rumus yang telah dijelaskan di atas. bila simpul i tidak memiliki anak yang dapat diturunkan(node daun) maka akan kembali ke langkag 2.
5. Untik setiap simpul anak yang diturunkan pada langkah 4 akan dimasukan ke dalam *priority queue.*
6. Kembali ke langkah 2.

Sementara untuk algoritma yang digunakan pada BFS adalah sama dengan yang telah dijelaskan pada bab sebelumnya karena tidak ada rumus yang perlu diubah pada algoritma BFS.

Berikut ini model dari pohon ruang status dalam pemencahan permasalahan rubik



Gambar 3.2 Pohon ruang status

Pada gambar hanya digunakan kotak dengan nomor karena kurang mampunya interface dalam menampilkan rubik yang mudah dan dapat dimengerti. Garis putus-putus pada gambar menggambarkan adanya status yang tidak tergambarkan.

Perbandingan waktu eksekusi antara algoritma BFS dan *Branch and bound* cukup terlihat dengan lebih cepatnya algoritma Branch and bound dalam menyelesaikan masalah ini sedangkan algoritma BFS membutuhkan waktu yang cukup lama untuk menyelesaikan permasalahan puzzle ini. Hal ini diakibatkan pada algoritma memeriksa seluruh ruang statu yang mana sangat banyak sehingga memakan waktu yang lama untuk menghasilkan rute solusi yang dicari. Sedangkan dalam Branch and bound memliki waktu yang lebih sedikit karena tidak seluruh status dibangkitkan dan hanya status yang memiliki *cost* yang kecil yang dibangkitkan. Perbedaan ini membuat waktu eksekusi program lebih sedikit dan perbedaan yang ditampilkan cukup signifikan.

IV. Kesimpulan

Dengan mengimplementasikan algoritma-algoritma ini dapat mengautomasi menyelesaian rubiks tetapi untuk lebih otptimal adalah algoritma *branch and bound* karena pada algoritma BFS memakan cukup banyak memori dan membuat komputer lambat dan waktu eksekusi yang juga lama.

REFERENCES

[1] https://ruwix.com/the-rubiks-cube/rubiks-cube-patterns-algorithms/ Diakses pada Kamis, 25 April 2019

[2] https://www.rubiks.com/about Diakses pada Kamis, 25 April 2019

[3] https://ruwix.com/the-rubiks-cube/rubiks-cube-patterns-algorithms/twisty-puzzle-patterns/ Diakses pada Kamis, 25 April 2019

[4] https://www.liputan6.com/citizen6/read/2680059/rekor-dunia-selesaikan-rubik-terpecahkan-473-detik Diakses pada Kamis, 25 April 2019

[5] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Branch-&-Bound-(2018).pdf Diakses pada Kamis, 25 April 2019

[6] https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/ Diakses pada Kamis, 25 April 2019

[7] https://em.wattpad.com/ead2932c9266933f06a578924899f3231ba7e822/68747470733a2f2f73332e616d617a6f6e6177732e636f6d2f776174747061642d6d656469612d736572766963652f53746f7279496d6167652f6b6859686131726b414d617a75513d3d2d3631383630383036332e31353462303332386630356532386333343133363333363738313933 2e706e67?s=fit&w=720&h=720 Diakses pada Kamis, 25 April 2019

[8] https://image.winudf.com/v2/image/Y29tLnJ1Yml4Q3ViZS5qb2hubG93ZGVyLnJ1Yml4Y3ViZV9pY29uXzBfOTdkZDEzZTE/icon.png?w=170&fakeurl=1 Diakses pada Kamis, 25 April 2019

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, 26 April 2019

Vijjasena / 13517084