Penerapan Algoritma BFS untuk Penyelesaian Kubus Rubik 2x2x2

Louis Cahyadi - 13517126
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
louiscahyadi@students.itb.ac.id

***Abstract*—Algoritma BFS merupakan salah satu algoritma yang sangat sering dipakai dalam penyelesaian masalah yang berkaitan dengan penelusuran graf dan pencarian lintasan terpendek. Selain itu kubus rubik merupakan salah satu permainan yang sangat terkenal dan sudah banyak dicoba oleh orang – orang dari berbagai kalangan. Permainan dengan tingkat kesulitan yang cukup tinggi ini dapat diselesaikan dengan penelusuran pada graf yang menggambarkan state dari rubik dengan menggunakan algoritma BFS.**

Kata kunci—BFS, graf, rubik

I. PENDAHULUAN

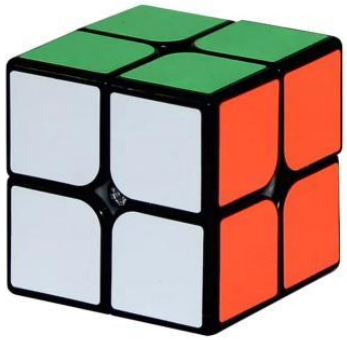
Kubus rubik merupakan permainan yang banyak dimainkan dan disukai orang – orang saat ini. Sulitnya menyelesaikan permainan ini menjadikan orang – orang tertarik untuk mencoba permainan ini. Selain itu kemajuan ilmu komputer memampukan manusia dapat melakukan perhitungan yang sulit dan besar secara cepat. Hal tersebut mendorong penggunaan komputer untuk melakukan pencarian solusi tercepat dengan langkah paling sedikit untuk menyelesaikan berbagai permainan, salah satunya ialah kubus rubik.

Salah satu algoritma yang banyak digunakan untuk menyelesaikan berbagai macam permainan ialah algoritma pencarian pada graf yaitu Breadth First Search (BFS) dan Depth First Search (DFS). Pencarian solusi dengan menggunakan algortima pada program komputer mampu menemukan langkah tercepat untuk menyelesaikan kubus rubik. Dalam makalah ini akan dilakukan penyelesaian kubus rubik dengan ukuran 2x2x2 dengan algoritma BFS

II. DASAR TEORI

A. *KUBUS RUBIK*

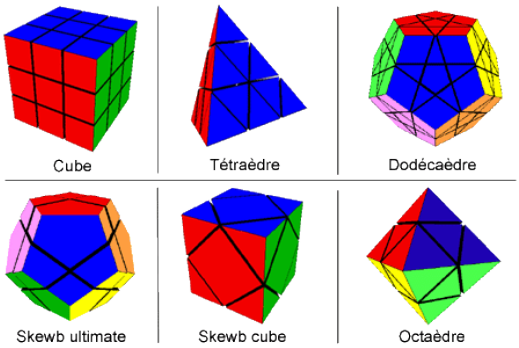
Kubus rubik merupakan sebuah permainan yang dibuat dari bahan sintetis. Terdiri dari berbagai macam bentuk dan ukuran. Pada kubus rubik 2x2x2, kubus terdiri dari 2x2x2 = 8 kubus – kubus kecil. Selain itu ada 6 x 4 muka kubus yang membentuk kubus 2x2x2. Rubik kubus terdiri dari 6 warna dengan masing masing wana muncul 4 kali. Kondisi atau state terakhir yang ingin dicapai dalam penyelesaian permainan ini adalah setiap sisi rubik hanya terdapat satu warna saja seperti pada gambar berikut ini



Gambar 1. Kubus rubik 2x2x2
(Sumber : www.blibli.com diakses pada 25 April 2019)

Kubus rubik dibuat dan dikembangkan oleh seorang Professor dari Budapest di Hungaria yang bernama Erno Rubik. Pada tahun 1974, Erno Rubik mengembangkan sebuah alat yang dapat digerakan berbentuk kubus untuk murid – muridnya di arsitektur sebagai media belajar 3 dimensi. Kubus buatan Erno Rubik merupakan rubik 3x3x3 yang sisi – sisinya dapat diputar – putar tanpa menyebabkan kerusakan pada kubus tersebut. Selain itu Erno Rubik juga menempelkan 54 stiker – stiker warna pada keenam sisi kubus.

Sejak saat itu permainan kubus rubik yang diciptakan oleh Erno Rubik diproduksi secara massal dan di sukai oleh orang – orang dari seluruh penjuru dunia. Bahkan dikembangkan variasi – variasi lain dari kubus tersebut seperti kubus rubik dengan ukuran 2x2x2, 4x4x4, bahkan ada bentuk – bentuk lain seperti piramid, balok, dll.



Gambar 2. Berbagai macam rubik
(Sumber : http://www.megasolver.com/ diakses pada 25 April 2019)

B. GRAPH

Graf G merupakan pasangan himpunan

$$G = (V, E)$$

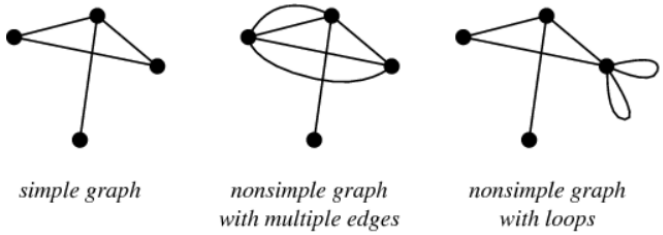
yang mana V merupakan himpunan titik – titik (simpul), dan E merupakan himpunan sisi – sisi yang titik – titik ujungnya berada pada himpunan V. Graf memiliki berbagai macam jenis berdasarkan sisinya, graf dapat dibedakan menjadi :

1. Graf Sederhana

Graf sederhana merupakan graf yang tidak mengandung gelang maupun sisi ganda

2. Graf tak-sederhana

Graf tak – sederhana merupakan graf yang mengandung gelang atau sisi ganda



Gambar 3. Graf sederhana, graf tak sederhana dengan sisi ganda, dan graf tak sederhana dengan gelang
(Sumber : http://mathworld.wolfram.com/SimpleGraph.html diakses pada 25 April 2019)

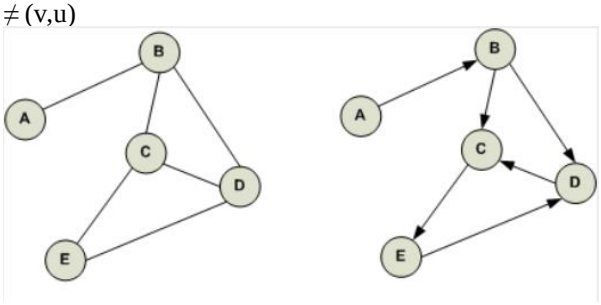
Selain itu, sisi dari sebuah graf dapat mengandung orientasi arah. Berdasarkan orientasi arah pada sisi graf, sebuah graf dapat dibedakan menjadi dua yaitu :

1. Graf tak – berarah

Graf tak berarah merupakan graf yang semua sisi – sisinya tidak memiliki orientasi arah. Pada jenis graf ini urutan pasangan simpul yang dihubungkan dengan sebuah sisi tidak diperhatikan, dengan kata lain (u, v) = (v, u)

2. Graf berarah

Graf berarah merupakan graf yang setiap sisinya memiliki orientasi arah sehingga urutan pasangan simpul yang dihubungkan oleh sebuah sisi diperhatikan, dengan kata lain (u,v) ≠ (v,u)



Gambar 4. Graf tak berarah dan graf berarah
(Sumber : https://algorithmsinsight.wordpress.com/graph-theory-2/ diakses pada 25 April 2019)

Pada penyelesaian permainan kubus 2x2x2 dengan algoritma bfs ini, digunakan graf sederhana dan graf tak-berarah, yang mana setiap simpulnya menyatakan state (kondisi) kubus rubik dan sebuah simpul terhubung dengan simpul yang lain jika dan hanya jika state dari suatu kubus rubik yang dinyatakan oleh simpul pertama dapat ditempuh lewat satu putaran pada kubus rubik, dan berlaku sebaliknya.

C. ALGORITMA BFS

BFS merupakan salah satu algoritma penelusuran pada sebuah graf. Dimulai dari sebuah titik s, dapat dicari sebuah lintasan ke sebuah titik yang diminta. Algoritma BFS juga dapat menyelesaikan persoalan penelusuran graf, dimulai dari sebuah titik, mengunjungi semua titik yang dapat dicapai pada graf.

Berbagai penerapan algoritma BFS dapat ditemukan pada hal – hal berikut ini :

1. Web Crawling

Melakukan penelusuran pada halaman – halaman web pada internet. Teknologi ini digunakan oleh search engine google.

2. Social Networking

Menyatakan hubungan pertemanan pada media sosial seperti Facebook, Twitter, dan Instagram.

3. Garbage Collection

Sistem manajemen memori pada bahasa pemrograman yang dapat mengidentifikasi dan menghapus isi dari memori – memori yang tidak lagi diakses oleh sebuah program.

4. Solving puzzles and games

Menyelesaikan berbagai macam puzzle dan permainan seperti rubik, permainan pluszle, dll.

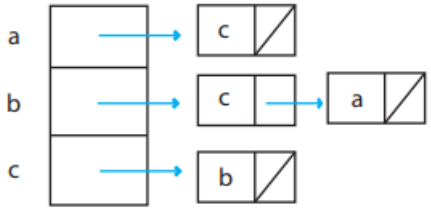
Untuk merepresentasikan graf pada program komputer dapat menggunakan berbagai macam jenis struktur data.

1. Matriks ketetanggaaan

Matriks yang baris dan kolom nya menyatakan nomor simpul dari graf tersebut. Dan entri dari baris ke i dan kolom ke j bernilai 0 jika simpul i dan simpul j tidak terhubung, dan bernilai 1 jikaa simpul i dan simpul j terhubung

2. List ketetanggaan

Menggunakan struktur data linked list, yang mana terdapat list yang menyatakan index dari simpul – simpul pada graf. Selanjutnya setiap simpul merupakan linked list yang berisi simpul – simpul yang dapat dikunjungi dari simpul tersebut.



Gambar 5. Linked list yang merepresentasikan sebuah graf

(Sumber : https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/MIT6_006F11_lec13.pdf diakses pada 25 April 2019)

3. Variasi object oriented

Menyatakan sebuah simpul sebagai objek dan memiliki atribut tetangga yang merupakan daftar simpul – simpul yang terhubung dengan simpul tersebut

Secara umum, BFS dapat dijelaskan sebagai sebuah algorima penelusuran graf yang menelusuri graf berdasarkan level – level ketetanggaan dari simpul awal s. Secara singkat algoritma BFS:

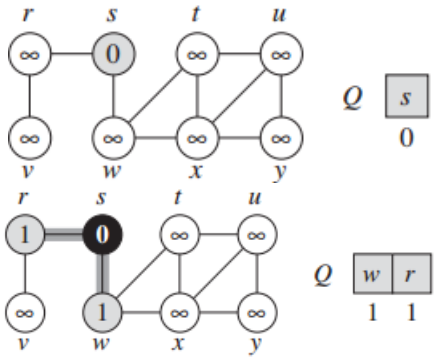
1. Kunjungi simpul v

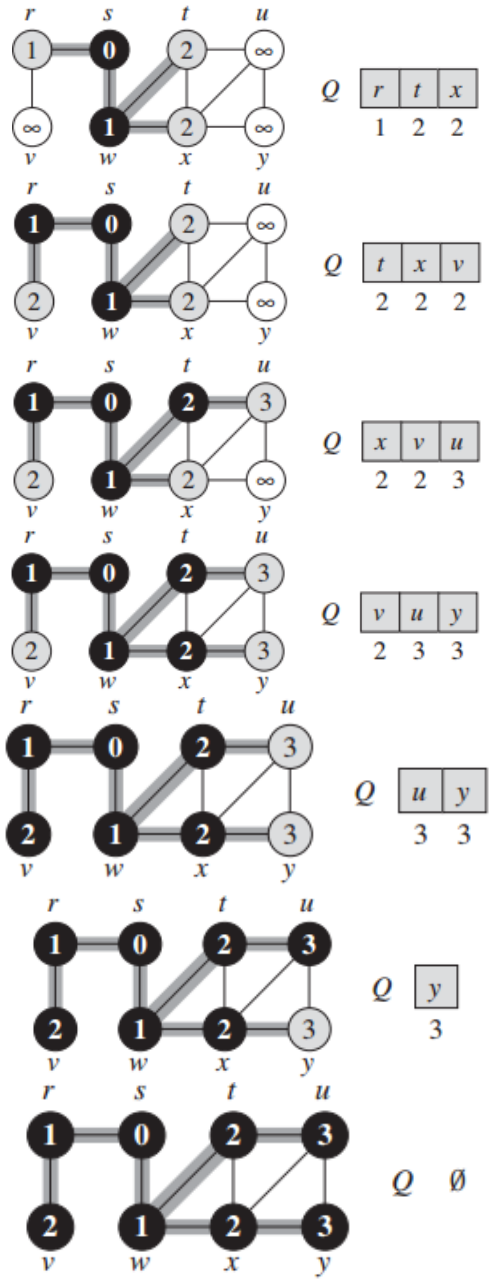
2. Kunjungi semua simpul tetangga dari v yang belum pernah dikunjungi sebelumnya.

3. Kunjungi simpul – simpul yang belum dikunjungi dan bertetangga dengan simpul – simpul yang sebelumnya telah dikunjungi pada poin 2.

Berikut ini merupakan pseudo-code dari algoritma BFS berdasarkan buku Introduction to Algorithm oleh Thomas H. Cormen third edition:

```
for (simpul u in G.V – {s})
        u.color = WHITE
        u.d = \infinite
        u.parent = NULL
s.color = GRAY
s.d = 0
s.parent = NULL
Q = {}
Q.put(s)
While (Q != {})
        u = Q.get()
        for (v in G.Adj[u])
                if v.color == WHITE
                        v.color = GRAY
                        v.d = u.d + 1
                        v.parent = u
                        Q.put(v)
        u.color = BLACK
```





Gambar 6. Ilustrasi penelusuran graf dengan menggunakan algoritma BFS
(Sumber : Introduction to Algorithm 3rd Edition by Thomas H. Cormen, etc.)

Secara umum algoritma BFS memiliki kompleksitas waktu O(|V| +| E|) dengan |V| menyatakan banyaknya simpul yang ada dan |E| menyatakan banyaknya sisi pada graf G. Selain itu dalam proses penelusuran graf terdapat 2 pendekatan yaitu pendekatan graf statis dan pendekatan graf dinamis.

Pada pendekatan statis, graf yang ditelusuri sudah terbentuk dari awal sebelum proses pencarian dimulai. Dan pada pendekatan graf dinamis, graf baru terbentuk selama proses pencarian dilakukan. Dengan pendekatan graf dinamis, graf tidak harus terbentuk seluruhnya ketika solusi ditemukan. Pada penyelesaian rubik 2x2x2 ini akan lebih baik

jika digunakan pendekatan graf dinamis, karena akan menghemat memori yang digunakan.

III. APLIKASI ALGORITMA BFS PADA PENYELESAIAN KUBUS RUBIK 2X2X2

A. Strukur Data

Warna – warna yang ada pada rubik dinyatakan dengan nomor 1 sampai 6. Sebuah state dari rubik didefinisikan oleh susunan warna dari ke-enam sisi rubik. Sehingga untuk menyimpan data state dari rubik digunakan struktur data list of list pada python, dengan contoh sebagai berikut

[[1,1,1,1], [2,2,2,2], [3,3,3,3], [4,4,4,4], [5,5,5,5], [6,6,6,6]]

Untuk menyimpan state – state dari rubik yang sudah pernah dikunjungi maka digunakan list dengan nama pernah_dikunjungi yang menampung state – state yang sudah pernah dikunjungi. Dan juga untuk menjalankan algoritma BFS dibutuhkan sebuah Queue dengan nama Daftar_BFS untuk menampung data statenya.

B. Fungsi dan Prosedur Pendukung

Untuk mensimulasikan perubahan state dari rubik jika dilakukan operasi operasi yang diperbolehkan dalam permainan dibuat fungsi/prosedur berikut

Gambar 7. Langkah – langkah perputaran yang mungkin dilakukan (Sumber : https://www.youcandothecube.com/solve-it/2-x-2-solution diakses pada 26 April 2019)

U (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian atas rubik ke kanan dari state old_configuration sebagai masukan.

Ui (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian atas rubik ke kiri dari state old_configuration sebagai masukan.

D (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian bawah rubik ke kanan dari state old_configuration sebagai masukan.

Di (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian bawah rubik ke kiri dari state old_configuration sebagai masukan.

F (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian depan rubik ke kanan dari state old_configuration sebagai masukan.

Fi (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian depan rubik ke kiri dari state old_configuration sebagai masukan.

B (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian belakang rubik ke kanan dari state old_configuration sebagai masukan.

Bi (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian belakang rubik ke kiri dari state old_configuration sebagai masukan.

L (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian kiri rubik ke kanan dari state old_configuration sebagai masukan.

Li (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian kiri rubik ke kiri dari state old_configuration sebagai masukan.

R (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian kanan rubik ke kanan dari state old_configuration sebagai masukan.

Ri (old_configuration) mengembalikan sebuah state dari rubik jika memutar bagian kanan rubik ke kiri dari state old_configuration sebagai masukan.

C. Implementasi Algoritma BFS

Pertama masukan terlebih dahulu state paling awal dari rubik ke Daftar_BFS serta ke list pernah_dikunjungi serta mendaftarkan parent dari state awal tersebut null.

Selanjutnya algoritma BFS dapat dijalankan sebagai berikut :

1. Ambil 1 simpul dari queue, misal diberi nama simpul_saat_ini jika simpul tersebut menyatakan state dari rubik yang sudah selesai maka BFS selesai.

2. Jika state tersebut bukan state akhir permainan maka, lakukan pengecekan pada kondisi jika dilakukan pemanggilan fungsi U apakah state yang dihasilkan sudah pernah dikunjungi. Jika pernah maka tidak dilakukan apapun, jika belum maka masukan state baru tersebut ke queue, list pernah_dikunjungi dan set parent dari state tersebut dengan state_saat_ini

3. Lakukan pemanggilan fungsi Ui pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

4. Lakukan pemanggilan fungsi D pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

5. Lakukan pemanggilan fungsi Di pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

6. Lakukan pemanggilan fungsi F pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

7. Lakukan pemanggilan fungsi Fi pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

8. Lakukan pemanggilan fungsi B pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

9. Lakukan pemanggilan fungsi Bi pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

10. Lakukan pemanggilan fungsi L pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

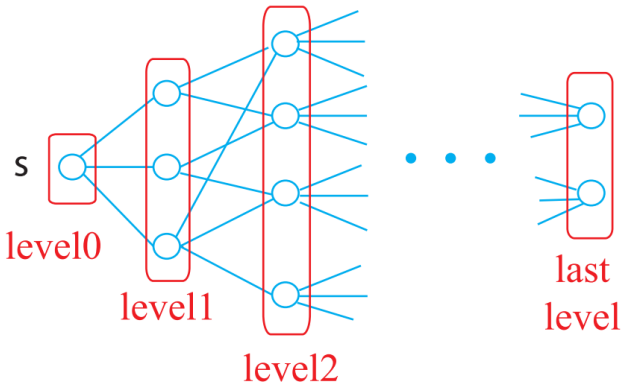
11. Lakukan pemanggilan fungsi Li pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

12. Lakukan pemanggilan fungsi R pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

13. Lakukan pemanggilan fungsi Ri pada state_saat_ini. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, list pernah_dikunjungi, dan set parent dari state tersebut dengan state_saat_ini

14. Lakukan kembali langkah 1 hingga 13 hingga didapat simpul yang merupakan simpul akhir permainan yaitu setiap sisi mempunyai warna yang sama, atau sampai size dari queue sudah menjadi 0 yang artinya kondisi awal rubik memang tidak dapat diselesaikan.

Dari langkah – langkah algoritma diatas, graf akan terbentuk menjadi terdiri dari beberapa level, yang bergantuk pada berapa langkah yang ditempuh dari state paling awal



Gambar 8. Level – level yang terbentuk dari algoritma BFS
(Sumber : https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/MIT6_006F11_lec13.pdf diakses pada 26 April 2019)

Jika algoritma BFS diatas sudah selesai dijalankan maka untuk mencari path atau langkah – langkah yang diambil dalam penyelesaian rubik dapat dilakukan dengan iterasi bergerak mundur dari state terakhir ke parentnya, dan parent dari parentnya dst, hingga ditemukan state paling awal.

IV. ANALISIS EKSPERIMEN

Pada saat penulis melakukan implementasi dari program diatas dengan menggunakan bahasa pemrograman python 3.6.7 pada sistem operasi Ubuntu 18.04 LTS untuk konfigurasi yang membutuhkan 3 atau kurang langkah maka hanya memakan waktu sekitar 1 detik. Untuk yang membutuhkan 4 langkah diperlukan waktu sekitar 10 detik. Dan untuk yang 5 langkah membutuhkan waktu hampir 5 menit.

Hal tersebut berbanding lurus dengan banyaknya simpul yang berada pada level tertentu (suatu simpul berada pada level yang sama dengan simpul yang lain jika dan hanya jika simpul tersebut dapat ditempuh dalam jumlah langkah yang sama dari simpul awal. Berdasarkan hitungan kasar, banyaknya simpul pada level ke L 11 kali lebih banyak dari banyaknya simpul pada level ke L – 1.

Penyelesaian persoalan rubik 2x2x2 ini dapat diselesaikan dengan lebih cepat dan efisien jika dapat mengoptimasi fungsi pengecekan apakah suatu state sudah pernah dikunjungi. Sebab ada banyak konfigurasi dari dua buah state yang terlihat berbeda padahal sama. Hal tersebut dapat mereduksi memori yang dibutuhkan dan mengurangi waktu yang dibutuhkan dalam menjalankan algoritma BFS.

V. KESIMPULAN DAN SARAN

Menyelesaikan permainan kubus rubik 2x2x2 dapat menerapkan algoritma BFS dalam mencari lintasan antara state awal dan state akhir dengan lintasan terpendek. Algoritma yang digunakan memiliki kompleksitas ruang O(V + E) dengan V menyatakan banyaknya simpul yang telah dibangkitkan dan E menyatakan banyaknya sisi yang ada pada graf, yang dalam kasus ini V ≈ 11^k dengan k merupakan ketinggian dari graf yang akan terbangun dan E ≈ 6 x V = 6 x 11^k

Kompleksitas ruang dan kompleksitas waktu dari algoritma yang digunakan pada makalah ini dapat dioptimasi dengan cara mengoptimasi fungsi pengecekan apakah dua buah state dari rubik merupakan keadaan yang sebenarnya sama dari rubik.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih yang sebesar – besarnya kepada Tuhan Yang Maha Esa, karena berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini. Penulis juga ingin mengucapkan terima kasih kepada seluruh pihak yang telah membantu penulis dalam menyelesaikan makalah ini. Terutama kepada seluruh dosen pengajar mata kuliah IF2211 Strategi Algoritma, Dr. Ir. Rinaldi Munir, M.T., Masayu Leylia Khodra, S.T., M.T., dan Dr. Nur Ulfa Maulidevi, S.T., M.Sc., yang telah berkenan membagikan ilmunya kepada seluruh peserta kuliah.

VII. REFERENSI

[1] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford.2009. “Introduction to Algorithms Third Edition”.MIT.

[2] Bandelow, Christoph.1982.”Inside Rubik’s Cube and Beyond”.Boston.Birkhauser.

[3] Prof. Demaine, Erik .2011.”Lecture 13: Graphs I : Breadth First Search”.MIT

[4] Munir, Rinaldi. 2006. “Diktat Kuliah IF2120 Matematika Diskrit”. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung.

[5] Tang, Adrian. 2008. “IMO Training 2008 : Graph Theory”. Canada.

[6] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/BFS-dan-DFS-(2019).pdf diakses pada 26 April 2019 pk 06.46 WIB

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

Louis Cahyadi, 13517126