Penerapan Algoritma *Breadth First Search* dalam Penyelesaian *Orientation Last Layer Parity* pada Rubik 4 x 4 x 4

Habibina Arif Muzayyan 13519125
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519125@std.stei.itb.ac.id

***Abstract*—Algoritma *Breadth First Search* (BFS) adalah salah satu algoritma untuk menyelesaikan masalah yang berkaitan dengan penelusuran graf dan pencarian lintasan terpendek. Sedangkan Rubik adalah permainan *puzzle* kombinasi warna 3 dimensi yang biasanya berbentuk kubus. Penyelesaian rubik berukuran besar (4 x 4 x 4, 5 x 5 x 5, dan seterusnya) biasanya menggunakan metode reduksi yaitu dengan menganggap rubik tersebut sebagai rubik 3 x 3 x 3 dan menyesuaikan warna pada tengah dan sisinya terlebih dahulu. *Orientation Last Layer* (OLL) *Parity* terjadi saat jumlah penyesuaian tengah atau sisinya tidak sama-sama genap atau ganjil akibatnya penyesuaian warna sisi yang terakhir terbalik. Permasalahan OLL *parity* ini dapat diselesaikan dengan penelusuran pada graf yang menggambarkan state dari rubik dengan menggunakan algoritma BFS.**

Keywords—BFS, Rubik, OLL Parity.

I. PENDAHULUAN

Rubik adalah permainan *puzzle* kombinasi warna 3 dimensi yang memiliki berbagai macam bentuk dan ukuran. Rubik diciptakan oleh Professor Arsitek Hungaria yang bernama Erno Rubik pada tahun 1974. Rubik pada awalnya hanya berbentuk kubus berukuran 3 x 3 x 3 tetapi mucul model-model baru yang memiliki ukuran yang berbeda seperti rubik 2 x 2 x 2, 4 x 4 x 4, 5 x 5 x 5, dan seterusnya. Bahkan beberapa rubik memiliki bentuk bukan kubus melainkan bentuk 3 dimensi lainnya.

Penyelesaian rubik kubus yang berukuran besar (4 x 4 x 4, 5 x 5 x 5, dan seterusnya) biasanya menggunakan metode reduksi yaitu dengan menyesuaikan warna pada tengah dan sisinya terlebih dahulu sehingga rubik tersebut dapat dianggap sebagai rubik 3 x 3 x 3. Namun, pada saat penyesuaian tengah atau sisinya dapat berjumlah tidak sama-sama genap atau ganjil sehingga menyebabkan kasus *parity* pada rubik tersebut. *Parity* adalah kondisi rubik ketika ada satu sisi yang warnanya terbalik dengan sisi yang lain. Salah satu *parity* yang sering terjadi adalah *Orientation Last Layer* (OLL) *Parity* yaitu sisi yang terakhir yang terletak di lapisan atas memiliki susunan warna yang terbalik.

Gambar 1.1 OLL *parity* pada rubik 4 x 4 x 4
(Sumber: www.kewbz.co.uk)

Terdapat beberapa algoritma yang dapat menyelesaikan masalah yang berkaitan dengan penelusuran graf dan pencarian lintasan terpendek. Salah satunya adalah algoritma *Breadth First Search* (BFS). Dengan merepresentasikan state rubik sebagai graf, maka permasalahan OLL *parity* ini dapat diselesaikan dengan penelusuran pada graf dengan menggunakan algoritma BFS.

II. DASAR TEORI

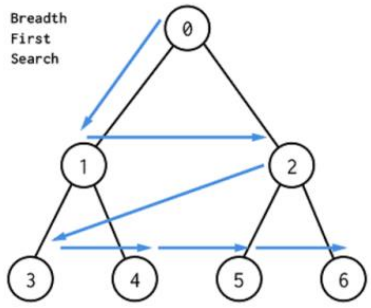
A. *Graf*

Graf merupakan struktur data diskrit yang mempunyai 2 komponen utama yaitu simpul (*node/vertice*) yang menyatakan objek dan sisi (*edge*) yang menyatakan hubungan antara 2 objek. Graf dapat didefinisikan sebagai notasi $G = (V, E)$ dengan V (*vertices*) menyatakan himpunan tak kosong dari simpul-simpul $\{v_1, v_2, \ldots, v_n\}$ dan E (edges) menyatakan himpunan sisi yang menghubungkan sepasang simpul $\{e_1, e_2, \ldots, e_n\}$. Perhatikan bahwa V adalah himpunan tak kosong dan E adalah himpunan boleh kosong. Akibatnya, suatu graf tidak memungkinkan hanya memiliki sisi, tetapi harus memiliki minimal satu simpul. Graf yang hanya terdiri dari satu simpul dan tidak memiliki sisi dinamakan dengan graf trivial.

Graf traversal adalah kumpulan algoritma atau cara sistematik untuk menelusuri simpul-simpul dari graf sesuai dengan kebutuhan. Graf traversal berguna untuk mencari solusi dari masalah yang direpresentasikan dengan graf. Terdapat dua jenis pendekatan graf, yaitu graf statis dan dinamis. Pada pendekatan graf statis, graf direpresentasikan sebagai struktur data dan sudah terbentuk sebelum proses pencarian. Sedangkan, graf dinamis dibangun selama pencarian solusi. Untuk mengemat penggunaan memori, penulis menggunakan pendekatan graf dinamis dengan menggunakan algoritma *Breadth First Search* (BFS) pada makalah ini.

B. Breadth First Search

Algoritma *Breadth First Search* (BFS) atau sering disebut pencarian melebar adalah algoritma penelusuran graf tanpa informasi yang mengunjungi simpul secara melebar. Penelusuran graf pada BFS dimulai dari simpul awal dan mengunjungi simpul-simpul yang bertetangga dengan simpul tersebut terlebih dahulu sebelum menuju ke level kedalaman berikutnya. Setiap simpul hanya dikunjungi tepat sekali saja atau dengan kata lain simpul yang sudah pernah dikunjungi tidak bisa dikunjungi lagi.



Gambar 2.1 Ilustrasi penelusuran graf dengan BFS
(Sumber: www.programmersought.com)

. Dalam implementasinya, BFS menggunakan struktur data matriks ketetanggaan untuk merepresentasikan struktur data graf, *queue* untuk membantu 'mengingat' simpul yang akan dikunjungi, dan *array* dengan nilai *boolean* untuk 'mengingat' simpul yang sudah dikunjungi. BFS memiliki kompleksitas waktu $O(|V| + |E|)$ dengan V adalah jumlah simpul dan E jumlah sisi. BFS lebih optimal dalam mencari lintasan terpendek dan lebih cocok untuk mencari simpul yang dekat dengan pusat.

Berikut ini adalah algoritma BFS secara umum dengan traversal yang dimulai dari simpul v.

1. Buat *queue* kosong
2. Masukkan simpul awal ke *queue*
3. Ubah nilai *boolean* dalam *array* yang berkoresponden dengan simpul awal menjadi *true*
4. Selama *queue* tidak kosong
   4.1. Hapus simpul pertama *queue*
   4.2. Untuk setiap simpul yang bertetangga dengan simpul pertama *queue*, jika nilai *boolean* dalam *array* yang berkoresponden dengan simpul tersebut adalah *false*, maka masukkan simpul tersebut ke *queue* dan ubah nilai *boolean* dalam *array* yang berkoresponden dengan simpul tersebut menjadi *true*

Berikut ini adalah notasi algoritmik dari BFS.

```
procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.

  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke layar
}
Deklarasi
   w : integer
   q : antrian;

   procedure BuatAntrian(input/output q : antrian)
   { membuat antrian kosong, kepala(q) diisi 0 }

   procedure MasukAntrian(input/output q:antrian, input v:integer)
   { memasukkan v ke dalam antrian q pada posisi belakang }

   procedure HapusAntrian(input/output q:antrian,output v:integer)
   { menghapus v dari kepala antrian q }

   function AntrianKosong(input q:antrian) → boolean
   { true jika antrian q kosong, false jika sebaliknya }

Algoritma:
   BuatAntrian(q)          { buat antrian kosong }

   write(v)                { cetak simpul awal yang dikunjungi }
   dikunjungi[v]←true      { simpul v telah dikunjungi, tandai dengan
                             true}
   MasukAntrian(q,v)       { masukkan simpul awal kunjungan ke dalam
                             antrian}

  { kunjungi semua simpul graf selama antrian belum kosong }
   while not AntrianKosong(q) do
        HapusAntrian(q,v)   { simpul v telah dikunjungi, hapus dari
                              antrian }
        for tiap simpul w yang bertetangga dengan simpul v do
              if not dikunjungi[w] then
                 write(w)     {cetak simpul yang dikunjungi}
                 MasukAntrian(q,w)
                 dikunjungi[w]←true
              endif
        endfor
   endwhile
  { AntrianKosong(q) }
```

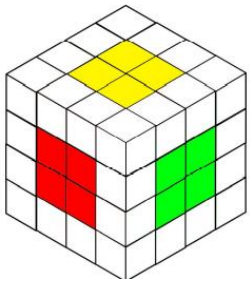
Gambar 2.2 Notasi Algoritmik BFS
(Sumber: informatika.stei.itb.ac.id)

C. Parity pada Penyelesaian Rubik 4 x 4 x 4

Terdapat banyak metode untuk menyelesaikan rubik 4 x 4 x 4. Salah satu metode yang sering digunakan dalam kompetisi rubik adalah metode yang digunakan oleh Yau, seorang pemegang rekor rubik. Metode yang digunakan oleh Yau ketika mengikuti kompetisi biasanya adalah turunan dari metode reduksi. Prinsip metode reduksi adalah dengan menganggap rubik 4 x 4 x 4 sebagai rubik 3 x 3 x 3 dengan menyesuaikan warna pada tengah dan sisinya terlebih dahulu.

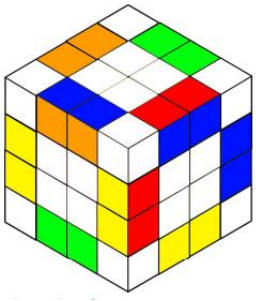
Secara umum, penyelesaian rubik 4 x 4 x 4 dengan metode reduksi adalah sebagai berikut.

1. Sesuaikan semua bagian tengah rubik pada setiap bidang, dan pastikan susunan warna pada setiap bidang sudah sesuai dengan urutannya yaitu warna putih berkebalikan dengan warna kuning, warna merah berkebalikan dengan warna oranye, dan warna biru berkebalikan dengan warna hijau.



Gambar 2.3 Penyesuaian bagian tengah rubik 4 x 4 x 4

2. Sesuaikan semua bagian sisi dengan sisi tetangga koresponden yang memiliki warna yang sama dengan sisi tersebut. Jumlah sisi yang harus disesuaikan adalah sebanyak 12 pasang.

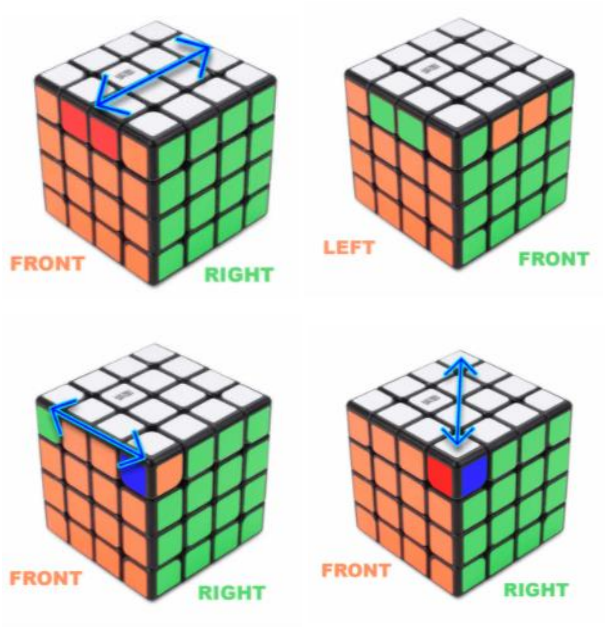


Gambar 2.4 Penyesuaian bagian sisi rubik 4 x 4 x 4

3. Karena semua bagian tengah dan sisi rubik sudah sesuai, sekarang rubik sudah direduksi ukurannya seperti rubik 3 x 3 x 3. Penyelesaiannya selanjutnya sama seperti penyelesaian rubik 3 x 3 x 3 yaitu menggunakan metode Fridrich.

Ketika penyelesaian rubik 4 x 4 x 4 dengan metode reduksi, terdapat kasus yang tidak ada pada penyelesaian rubik 3 x 3 x 3 yaitu kasus *parity*. *Parity* dapat terjadi saat penyesuain warna pada sisi rubik yang tidak sama-sama ganjil atau genap. *Parity* mengakibatkan susunan warna pada beberapa sisi rubik salah posisi atau bahkan terbalik urutannya. Pada penyelesaian rubik 4 x 4 x 4, terdapat dua jenis *parity* yaitu *Permutation Last Layer* (PLL) *parity* dan *Orientiation Last Layer* (OLL) *Parity*.

PLL *parity* adalah ketika susunan salah satu sisi atau sudut rubik salah posisi. PLL *parity* terdiri dari 4 kasus berdasarkan kesalahan posisinya.



Gambar 2.5 PLL *parity* pada rubik 4 x 4 x 4
(Sumber: www.kewbz.co.uk)

OLL *parity* adalah ketika susunan salah satu sisi rubik terbalik. OLL *parity* hanya ada satu kasus pada penyelesaian rubik 4 x 4 x 4.

III. Implementasi

Pada Implementasi kali ini, penulis menggunakan Bahasa Python. Program menggunakan beberapa struktur data yaitu *list* untuk mengimplementasikan *queue*, dan simpul, Program juga menggunakan struktur data *map* untuk mengimplementasikan graf dan *array* dikunjungi. Program dibantu dengan beberapa fungsi dan prosedur pendukung agar program berjalan dengan efektif.

A. Struktur Data

Penulis mengimplementasikan graf berupa *map* dengan *key* adalah simpul dan *value* adalah *parent* dari simpul. Penulis mengimplementasikan *queue* berupa *list* untuk membantu 'mengingat' simpul yang akan dikunjungi. Penulis mengimplementasikan struktur data *map* untuk 'mengingat' simpul yang sudah dikunjungi dengan *key* berupa simpul dan *value* berupa *boolean*.

Untuk merepresentasikan state rubik sebagai sebuah simpul graf, penulis mengimplemntasikan struktur datanya berupa *list of list* dengan mensubsitusikan warna putih sebagai karakter **w**, merah sebagai karakter **r**, biru sebagai karakter **b**, kuning sebagai karakter **y**, oranye sebagai karakter **o**, dan hijau sebagai karakter **g**. Perspektif rubik yang digunakan dalam program adalah bidang warna putih di atas, merah di belakang, biru di kiri, kuning di bawah, oranye di depan, dan hijau di kanan.

State awal rubik berupa kondisi ketika terjadi OLL *parity* yaitu susunan warna salah satu sisi putih oranye terbalik. Dan state akhir rubik berupa kondisi rubik dengan warnanya sudah tersusun sempurna.

Struktur data dari state awal adalah sebagai berikut:

```
awal = [
    ['w','w','w','w',
     'w','w','w','w',
     'w','w','w','w',
     'w','o','o','w'],
    ['r','r','r','r',
     'r','r','r','r',
     'r','r','r','r',
     'r','r','r','r'],
    ['b','b','b','b',
     'b','b','b','b',
     'b','b','b','b',
     'b','b','b','b'],
    ['y','y','y','y',
     'y','y','y','y',
     'y','y','y','y',
     'y','y','y','y'],
    ['o','w','w','o',
     'o','o','o','o',
     'o','o','o','o',
     'o','o','o','o'],
    ['g','g','g','g',
     'g','g','g','g',
     'g','g','g','g',
     'g','g','g','g']]
```

Struktur data dari state akhir adalah sebagai berikut:

```
akhir = [
    ['w','w','w','w',
     'w','w','w','w',
     'w','w','w','w',
     'w','w','w','w'],
    ['r','r','r','r',
     'r','r','r','r',
     'r','r','r','r',
     'r','r','r','r'],
    ['b','b','b','b',
     'b','b','b','b',
     'b','b','b','b',
     'b','b','b','b'],
    ['y','y','y','y',
     'y','y','y','y',
     'y','y','y','y',
     'y','y','y','y'],
    ['o','o','o','o',
     'o','o','o','o',
     'o','o','o','o',
     'o','o','o','o'],
    ['g','g','g','g',
     'g','g','g','g',
     'g','g','g','g',
     'g','g','g','g']]
```

B. Fungsi dan Prosedur

Berikut ini adalah fungsi dan prosedur pada program untuk mensimulasikan perubahan state atau gerakan pada rubik 4 x 4 x 4.

1. R(state), mengembalikan sebuah state dari rubik jika memutar satu bagian luar kanan rubik searah jarum jam.
2. RAksen(state) mengembalikan sebuah state dari rubik jika memutar satu bagian luar kanan rubik berbalik arah jarum jam.
3. r(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam kanan rubik searah jarum jam.
4. rAksen(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam kanan rubik berbalik arah jarum jam..
5. L(state), mengembalikan sebuah state dari rubik jika memutar satu bagian luar kiri rubik searah jarum jam.
6. LAksen(state) mengembalikan sebuah state dari rubik jika memutar satu bagian luar kiri rubik berbalik arah jarum jam.
7. l(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam kiri rubik searah jarum jam.
8. lAksen(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam kiri rubik berbalik arah jarum jam.
9. F(state), mengembalikan sebuah state dari rubik jika memutar satu bagian luar depan rubik searah jarum jam.
10. FAksen(state) mengembalikan sebuah state dari rubik jika memutar satu bagian luar depan rubik berbalik arah jarum jam.
11. f(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam depan rubik searah jarum jam.
12. fAksen(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam depan rubik berbalik arah jarum jam.
13. B(state), mengembalikan sebuah state dari rubik jika memutar satu bagian luar belakang rubik searah jarum jam.

14. BAksen(state) mengembalikan sebuah state dari rubik jika memutar satu bagian luar belakang rubik berbalik arah jarum jam.
15. b(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam belakang rubik searah jarum jam.
16. bAksen(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam belakang rubik berbalik arah jarum jam.
17. U(state), mengembalikan sebuah state dari rubik jika memutar satu bagian luar atas rubik searah jarum jam.
18. UAksen(state) mengembalikan sebuah state dari rubik jika memutar satu bagian luar atas rubik berbalik arah jarum jam.
19. u(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam atas rubik searah jarum jam.
20. uAksen(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam atas rubik berbalik arah jarum jam.
21. D(state), mengembalikan sebuah state dari rubik jika memutar satu bagian luar bawah rubik searah jarum jam.
22. DAksen(state) mengembalikan sebuah state dari rubik jika memutar satu bagian luar bawah rubik berbalik arah jarum jam.
23. d(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam bawah rubik searah jarum jam.
24. dAksen(state), mengembalikan sebuah state dari rubik jika memutar satu bagian dalam bawah rubik berbalik arah jarum jam.

C. *Algoritma*

Berikut ini adalah algoritma dari program.

1. Buat *list* kosong untuk merepresentasikan *queue*, Buat *map* untuk merepresentasikan graf dengan *key* berupa simpul (state rubik) dan *value* berupa *parent*, dan Buat *map* untuk 'mengingat' simpul yang sudah dikunjungi.
2. Masukkan state awal ke dalam *queue*
3. Ubah nilai *boolean* dalam *map* yang berkoresponden dengan state awal menjadi *true*
4. Set *parent* dari state awal berupa null
5. Selama *queue* tidak kosong

5.1 Jika elemen pertama *queue* berupa state akhir, maka program selesai dijalankan

5.2 Jika tidak, maka untuk setiap fungsi yang mensimulasikan perubahan state atau gerakan pada rubik: lakukan pemanggilan fungsi tersebut dengan paramater berupa elemen pertama *queue*. Jika state baru yang dihasilkan belum pernah dikunjungi maka tambahkan state tersebut ke queue, ubah nilai *boolean* dalam *map* yang berkoresponden dengan state tersebut menjadi *true*, dan atur *parent* dari state yang dihasilkan berupa elemen pertama *queue*, serta hapus elemen pertama *queue*.

6. Lakukan iterasi mundur dari state akhir ke state awal untuk mendapatkan solusi penyelesaiannya

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

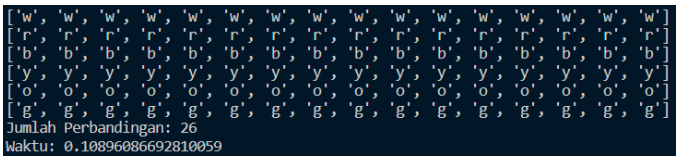
IV. Analisa

Dengan menggunakan algoritma ini, program 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 $V \approx 24^k$ dengan k merupakan ketinggian dari graf yang akan terbangun dan $E \approx 6 \times V = 6 \times 24^k$.

Pada saat penulis mengimplementasikan program, penulis mencoba dengan state awal yang memiliki level kedalaman 2, seperti sebagai berikut:

```
awal = [
    ['w','w','w','y',
     'w','w','w','y',
     'w','w','w','y',
     'w','w','w','y'],
    ['o','r','r','r',
     'o','r','r','r',
     'o','r','r','r',
     'o','r','r','r'],
    ['b','b','b','b',
     'b','b','b','b',
     'b','b','b','b',
     'b','b','b','b'],
    ['y','y','y','w',
     'y','y','y','w',
     'y','y','y','w',
     'y','y','y','w'],
    ['o','o','o','r',
     'o','o','o','r',
     'o','o','o','r',
     'o','o','o','r'],
    ['g','g','g','g',
     'g','g','g','g',
     'g','g','g','g',
     'g','g','g','g']]
```

Didapat hasil sebagai berikut.



Berdasarkan KewbzUK, Solusi penyelesaian OLL *parity* adalah *r2 B2 U2 l U2 r' U2 r U2 F2 r F2 l' B2 r2*. Penyelesaian ini membutuhkan 25 langkah. Jika penyelesaian dicari menggunakan program pada makalah ini, maka setidaknya membutuhkan waktu yang sangat lama.

Karena perspektif rubik yang digunakan dalam program bersifat statis, maka ada beberapa state yang tampaknya berbeda tetapi susunan warnanya sama jika dilihat dari perspektif lain. Untuk mengatasi hal tersebut, program diharapkan dapat mengoptimasi fungsi pengecekan apakah suatu state sudah pernah dikunjungi berdasarkan perspektif lain sehingga mereduksi memori yang dibutuhkan dan mengurangi waktu yang dibutuhkan dalam menjalankan algoritma BFS. Namun, penulis belum sempat mengimplementasikan fungsi tersebut.

V. Simpulan dan Saran

Penyelesaian OLL *Parity* pada rubik 4 x 4 x 4 dapat menerapkan algoritma BFS tidak ditemukan karena keterbasan perangkat penulis dalam menjalankan program yang berat. Karena 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 $V \approx 24^k$ dengan k merupakan ketinggian dari graf yang akan terbangun dan $E \approx 6 \times V = 6 \times 24^k$. Pada makalah ini, k bernilai banyaknya gerakan rubik pada solusi penyelesaian yaitu 25.

Sayangnya, penyelesaian menggunakan algoritma BFS ini tidak cukup mangkus karena memerlukan kompleksitas ruang dan kompleksitas waktu yang sangat besar.

Untuk mereduksi memori yang dibutuhkan dan mengurangi waktu yang dibutuhkan dalam menjalankan program, sebaiknya ditambahkan fungsi pengecekan apakah suatu state sudah pernah dikunjungi berdasarkan perspektif lain sehingga kompleksitas ruang dan waktu yang digunakan diharapkan dapat berkurang.

Link Video Youtube

https://youtu.be/z8jVDprg8JY

Ucapan Terima Kasih

Penulis mengucapkan puji syukur dan terima kasih atas kehadirat Tuhan yang Maha Esa karena atas berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini yang berjudul "Penerapan Algoritma *Breadth First Search* dalam Penyelesaian *Orientation Last Layer Parity* pada Rubik 4 x 4 x 4" dengan baik. Kemudian, Penulis berterima kasih kepada Bapak Prof. Ir. Dwi Hendratmo Widyantoro, M.Sc., Ph.D. selaku dosen pengajar mata kuliah Strategi Algoritma di K03, yang telah menyediakan waktu dan tenaganya untuk menyampaikan ilmunya kepada mahasiswa di K03. Penulis juga berterima kasih kepada keluarga dan kerabat yang telah memberi dukungan kepada Penulis untuk menyelesaikan makalah ini.

Referensi

[1] Bandelow, Christoph. 1982. "Inside Rubik's Cube and Beyond". Boston. Birkhauser.

[2] P.B. Echeverria and J. Westermark. 2013. "Benchmarking Rubik's Revenge algorithms" Stockholm. Final Report.

[3] https://www.kewbz.co.uk/blogs/solutions/4x4-parity-algorithms-and-cases diakses pada 8 Mei 2021

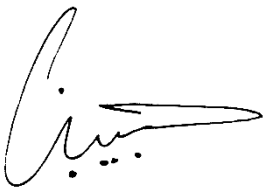
[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf diakses pada 10 Mei 2021

[5] https://www.programmersought.com/article/36944629579/ diakses pada 11 Mei 2021

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.

Jambi, 11 Mei 2021



Habibina Arif Muzayyan 13519125