Perbandingan Kecepatan Antara Algoritma Iterative Deepening A* dan Algoritma Branch And Bound dalam Puzzle 15

Mario Gunawan / 13518114[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13518114@std.stei.itb.ac.id

***Abstract*— Puzzle 15 adalah permainan bergenre puzzle. Cara memainkannya adalah dengan menyusun angka dari 1 sampai 15 secara acak tanpa duplikat ke dalam matriks 4x4, sehingga ada satu kotak yang kosong. Tujuan permainan adalah menggeser kotak yang kosong sedemikian rupa sehingga angka 1 sampai 15 berurutan dari kiri ke kanan, atas ke bawah, dan kotak di ujung kanan bawah kosong.**

Keywords— IDA*, Puzzle 15, Branch and Bound

I. PENDAHULUAN

Strategi algoritma adalah bidang informatika yang mempelajari tentang kecepatan dan keefektifan suatu algoritma dibanding algoritma lainnya. Algoritma yang digunakan dinyatakan baik apabila algoritma itu cepat dan tidak memakan banyak memori, sebaliknya, algoritma yang digunakan dinyatakan kurang baik apabila algoritma itu tidak cepat dan memakan banyak memori. Contoh algoritma yang dipelajari strateegi algoritma adalah *Brute Force, Divide and Conquer, A** dan *Backtracking.* Algoritma yang digunakan pada makalah ini adalah Iterative Deepening A*(IDA*) dan Branch and Bound untuk membandingkan kecepatan algoritmanya dalam menyelesaikan permainan puzzle 15.

Kedua algoritma yang digunakan dalam makalah ini merupakan algoritma pencarian yang tergolong cepat dan mangkus untuk menyelesaikan masalah pencarian. Namun, tetep ada kekurangan dan kelebihan masing masing algoritma, dan itu yang akan dibahas dalam makalah ini dengan puzzle 15 sebagai perbandingan.

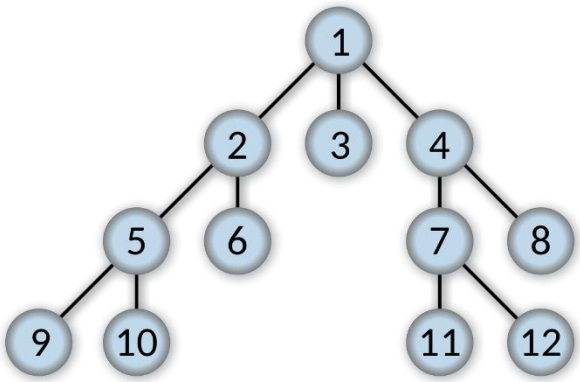
Untuk program puzzle 15 branch and bound yang dipakai adalah program dengan algoritma yang diajarkan kepada kami di STEI ITB dan telah kami buat pada tugas sebelumnya. Sementara program penyelesaian puzzle IDA* menggunakan program luar yang didapat dari https://codegolf.stackexchange.com/questions/6884/solve-the-15-puzzle-the-tile-sliding-puzzle (ref: [5]) , dari jawaban username bernama “orlp”.

II. DASAR TEORI

II.1 Algoritma Branch and Bound

Algoritma Branch and Bound merupakan algoritma yang digunakan untuk persoalan optimisasi, yaitu mencari solusi yang paling minimum atau paling maksimum dari suatu persoalan. Algoritma ini digunakan untuk mencari suatu solusi dari persoalan.

Metode yang digunakan dalam algoritma ini mirip dengan algoritma BFS (bradth first search) , sebuah algoritma yang mengeksplor semua kemungkinan solusi dari suatu persoalan satu per satu dari akar persoalan ke solusi, dan dilakukan bertahap (mengeksplor kemungkinan solusi yang sejajar terlebih dahulu), contoh:



Gambar II-1 Cara Pemecahan Algoritma Branch And Bound
sumber:https://upload.wikimedia.org/wikipedia/commons/thumb/3/33/Breadth-first-tree.svg/1200px-Breadth-first-tree.svg.png

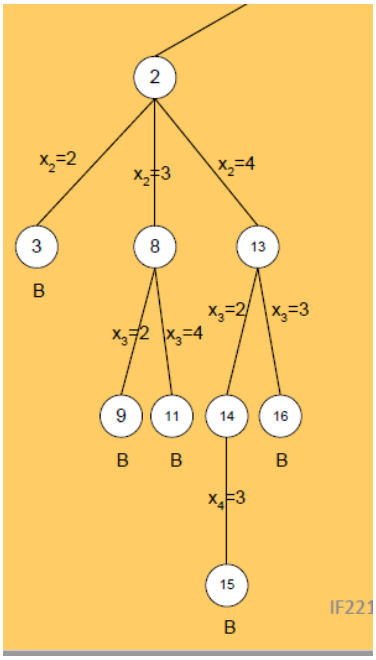
Suatu persoalan bisa memiliki banyak kemungkinan arah yang bisa diambil untuk mendapatkan solusi. Misalnya di gambar 2.1, persoalan awal yang diberi nomor 1 memiliki banyak cabang cara penyelesaian. Contoh dalam dunia nyata adalah cara menyelesaikan sudoku. Cara pencarian suatu angka dalam sudoku bisa dengan melihat satu baris, satu kolom, maupun satu kotak. Kita tidak tahu pasti yang mana yang akan mengarah ke jawaban bila kita tidak punya “data” atau pengalaman yang

membuat kita tahu langkah mana yang harus diambil. Sama seperti komputer, bila tidak ada data yang memudahkan suatu pencarian, maka komputer tidak akan tahu mana arah yang harus dicari agar mendapat solusi.

Cara penyelesaian BFS adalah dengan satu megeksplor satu "level" sampai habis, baru melanjutkan ke "bawah" (ke simpul yang belum dieksplor) seperti gambar diatas, angka 1 – 12 menyatakan urutan pncarian dilakukan. tiap level dihabiskan dulu mencarinya , baru lanjut ke level dibawahnya sampai ditemukan solusi.

Cara BFS dinilai kurang cepat apabila ada banyak sekali kemungkinan arah dari suatu program, sehingga solusi akan lama didapat karena algoritma ini akan mengeksplor tiap cabang yang belum dieksplor. Oleh karena lamanya algoritma BFS, maka digunakanlah Branch and Bound(branch and bound) untuk menyelesaikan masalah eksplorasi cabang ini.

Algoritma Branch and Bound memanfaatkan pengetahuan akan "harga" dari suatu simpul yang dieksplorasi terlebih dahulu, sehingga simpul dengan harga yang mahal tidak akan dieksplorasi, sampai simpul tersebut merupakan simpul dengan harga termurah dari set semua simpul yang belum dieksplorasi. Contohnya sebagai berikut:



Gambar II-2 Pohon Algoritma Branch and Bound
sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/stima19-20.htm slide Algoritma branch and bound

Pada contoh di atas, setiap simpul memiliki cost sesuai dengan angka yang ada pada simpul terseebut. Pada contoh diatas, branch and bound mencari simpul dengan harga maksimal, maka bila didapat simpul yang belum dieksplorasi dan harganya lebih kecil atau sama dengan dari harga maksimal saat ini, simpul tersebut tidak dieksplorasi dahulu sampai nanti ketika simpul tersebut mempunyai harga tertinggi dan belum ditemukan solusi.

Bila didapat simpul dengan solusi, maka stop dan nyatakan harga simpul tersebut sebagai harga optimum dari persoalan. Algoritma ini adalah pencarian dengan algoritma greedy by cost.

Secara umum, cara menghitung suatu cost dari sebuah simpul X akan dihampiri dari simpul Y adalah :

1. Menghitung "kedalaman" dari simpul akar ke simpul X (berapa *level* yang dilalui) , kita namai kedalaman depth
2. Menghitung "harga" dari simpul Y ke simpul X menggunakan fungsi yang didefinisikan secara eksplisit di algoritma, kita namakan hargaYkeX
3. Harga dari simpul X , kita namakan hargaTotal adalah:

```
hargaTotal = depth + hargaYkeX
```

Pada Puzzle 15 yang Penulis bahas, penghitungan harga X ke Y adalah sebagai berikut :

```
hargaXkeY = banyak angka yang tidak ada dalam posisi final matriks tidak termasuk balok kosong
```

Misalnya :

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

Gambar II-3 Contoh matriks puzzle 15
sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/stima19-20.htm slide Algoritma branch and bound

Pada matriks diatas, hargaXkeY adalah 3, karena angka 7, 12, dan 11 tidak dalam posisi yang benar. Algoritma Branch and Bound untuk persoalan ini menggunakan harga yang minimum untuk dieksplor terlebih dahulu.

II.2 **Algoritma Iterative Deepening A***

A. Algoritma A*

Algoritma A* adalah algoritma yang mirip dengan algoritma branch and bound. Yaitu meminimilisasi estimasi harga dari suatu persoalan dengan cara menelusuri simpul yang dianggap paling dekat menuju goal. Perbedaan utamanya adalah fungsi penghitungan harga dari suatu simpul.

Pada algoritma A*, penghitungan harga suatu simpul X yang dengan simpul tujuan simpul Z dilakukan dengan rumus sebagai berikut :

1. Menghitung harga dari akar ke simpul X, berbeda dari Branch and Bound yang menghitung kedalaman., kita namai g(n)
2. Menghitung **estimasi** harga dari simpul X ke simpul Z. Estimasi menggunakan fungsi yang

didefinisikan secara eksplisit di algoritma. Perhatikan bahwa dalam Branch and Bound, tahap keduanya adalah mengihitung harga dari simpul sebelumnya ke simpul X. Kita namai h(n) atau fungsi heuristik

3. Maka, harga total simpul X, kita namakan f(n), adalah:

```
f(n) = g(n) + h(n)
```

B. Algoritma Iterative Deepening A*

1. Algoritma DFS

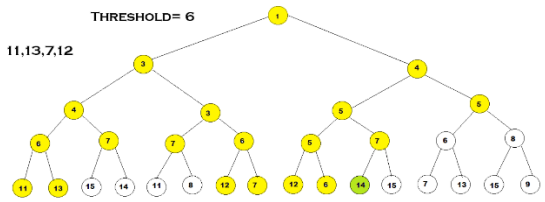
Algoritma *deep first search*, biasa dikenal sebagai algoritma DFS, adalah algoritma pencarian yang mencari suatu simpul *goal* dari simpul akar dengan cara menelusuri simpul yang baru saja didapat. Karena cara penelusurannya ini, DFS akan terus mencari dalam satu simpul sampai akarnya semua sudah ditelusuri, lalu pindah ke simpul selanjutnya sehingga pencarian akan sampai level yang dalam.

2. Algoritma IDS

Algoritma *Iterative Deepening Search*, biasa dikenal sebagai algoritma IDS, adalah algoritma variasi dari DFS yang mencari suatu simpul *goal* dari simpul akar dengan cara pengulangan dari akar menuju simpul dengan limit kedalaman tertentu. Maksud kedalaman adalah menggunakan batasan level, sehingga tidak akan mengeksplor sampai kedalaman yang infinit. Perbedaan utama dengan BFS (ref : [6]) adalah IDS mengulangi pencariannya dari awal, sementara BFS menyimpan *state* dari seluruh pencarian. Kecepatan IDS dengan BFS susah diukur, karena terkadang satu lebih cepat dari yang lain, namun penggunaan memori IDS lebih sedikit daripada BFS.

Algoritma IDA* adalah algoritma yang menggunakan fungsi penghitungan harga A* namun eksplorasi simpulnya menggunakan batasan dari algoritma iterative deepening search(IDS).

Algoritma IDA* mengeksplor per level layaknya IDS, namun perbedaannya adalah, pembatas dari eksplorasi simpul bukanlah level, namun fungsi f(n) dalam algoritma A*. Algoritma akan dimulai dari akar, dengan f(n) tertentu dan disimpan kedalam variabel yang kita namakan pembatas, dan bila suatu node ditemukan dengan f(n) yang lebih besar dari pembatas, maka pencarian di node tersebut akan dihentikan, dan nilai f(n) node tersebut disimpan ke dalam senarai kandidat pembatas . Bila suatu node ditemukan sebagai goal, stop, kalau bukan goal dan f(n) kurang dari pembatas, maka teruskan pencarian di node itu layaknya DFS sampai ditemukan goal, atau f(n) dari semua cabangnya lebih besar dari pembatas. Bila belum ketemu goal dan semua cabang sudah berhenti dicari (f(n) masing masing cabang lebih besar dari pembatas), maka ulangi pencarian dengan pembatas = nilai terkecil dari senarai kandidat pembatas, lakukan terus sampai ketemu goal. Contoh ilustrasi berupa gif dapat dilihat di halaman ini:

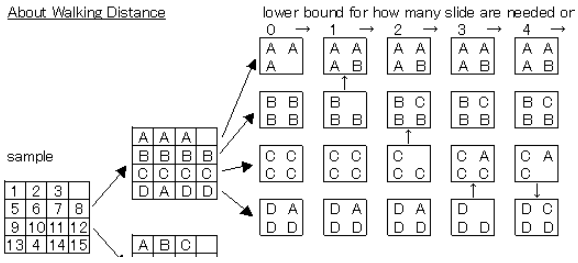


Gambar II-4 Ilustrasi algoritma IDA (GIF), dapat dengan membuka sumber*
sumber:
https://algorithmsinsight.files.wordpress.com/2016/03/ida-star.gif?w=714&zoom=2

Pada puzzle 15, fungsi g(n) sama seperti branch and bound, yaitu kedalaman dari akar sampai ke simpul tersebut (level), sementara h(n) sebagai pembatas yang digunakan digunakan dengan algoritma yang dinamakan *walking distance*, dibuat oleh Ken'ichiro Takahashi dari Jepang. Algoritma untuk fungsi heuristik ini menggunakan memiliki tahap sebagai berikut:

1. Ubah matriks puzzle, buat 2 matriks baru yang semua elemennya sama dengan matriks puzzle, lalu untuk matriks pertama, tandai angka yang seharusnya ada di baris pertama sebagai A, baris kedua sebagai B, dan seterusnya. Untuk matriks kedua, tandai angka yang seharusnya ada di kolom pertama sebagai A, kolom kedua sebagai B, dan seterusnya.
2. Bagi kedua matriks 4x4 yang didapat menjadi 4 matriks 2x2 yang salah satu tabel mengandung elemen baris, atau yang lainnya elemen kolom . Urutan tidak dipermasalahkan.
3. Pindahkan satu per satu elemen matriks 2x2 keatas dan kebawah untuk tabel yang mengandung baris, kekiri dan kekanan untuk tabel yang mengandung kolom sampai semua matriks dalam satu tabel berisi satu elemen yang seragam, dan berurut.
4. Hitung jumlah langkah sampai didapatkan matriks 2x2 seragam dan terurut.
5. Jumlahkan jumlah tahap kedua matriks 4x4. Jumlah tahap adalah nilai *walking distance*.

Untuk lebih jelasnya, coba lihat



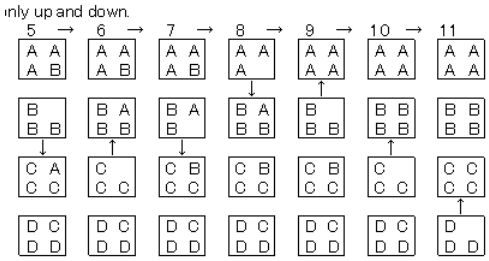
Gambar II-5 Walking distance algorithm
sumber: http://www.ic-net.or.jp/home/takaken/e/15pz/wd.gif

Seperti pada langkah pertama, matriks sample merupakan matriks awal, lalu dibuat 2 matriks, matriks yang ditunjukkan pada gambar diatas merupakan matriks baris, sehingga 1 menjadi A, 2 menjadi A, dan seterusnya. Angka empat yang berada dibawah juga menjadi A.

Lalu pada tahap kedua, bagi matriks baris menjadi 4

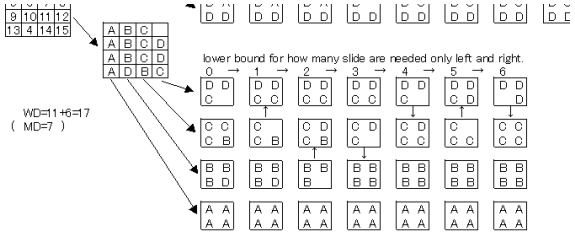
matriks 2x2 yang isinya elemen baris matriks baris. Ingat, urutan tidak penting.

Tahap ketiga, coba selesaikan posisi matriks 2x2 dengan menukar hanya keatas dan kebawah setiap matriks 2x2, urutan tidak penting, sampai didapat semua matriks elemennya sesuai (4 A di matriks pertama, 4 B di matriks kedua, 4 C di matriks ketiga, 3 D di matriks ke 4).



Gambar II-6 Penyelesaian matriks baris
sumber: http://www.ic-net.or.jp/home/takaken/e/15pz/wd.gif

Lalu tahap terakhir, hitung jumlah tahap yang harus dilakukan. Dalam sampel ini, matriks baris memiliki 11 tahap. dan matriks kolom memiliki 6 tahap. Tahapan matriks kolom ada di Gambar II-7.



Gambar II-7 Penyelesaian matriks kolom
sumber: http://www.ic-net.or.jp/home/takaken/e/15pz/wd.gif

Hitung jumlah langkah total yaitu 6 + 11 = 17. Walking distance untuk sampel ini adalah 17. Bila sampel ada di level 8, maka fungsi f(n) bernilai 25 (didapat dari 17 + 8). Dapat dilihat bahwa bila h(n) bernilai 0, maka didapatkan goalnya.

III. DESKRIPSI PERSOALAN

Masalah yang sering kali menjadi tema dalam mata kuliah strategi algoritma adalah masalah kecepatan (efisiensi) sebuah algoritma atau masalah space (memori) yang diperlukan. Contoh masalah kecepatan adalah algoritma bubble sort dengan algoritma quick sort. Keduanya mangkus, namun bubble sort memakan waktu yang jauh lebih lama untuk mendapatkan solusi terutama bila elemen senarai banyak, sementara quicksort memerlukan waktu yang jauh lebih sedikit, dan semakin bertambahnya elemen senarai, semakin terlihat perbedaan kecepatan quicksort dan bubblesort.

Masih mengenai sorting, algoritma bubble sort lebih lamban dari quicksort, namun algoritma quick sort memerlukan space di memori sebanyak O(log n) , artinya seiring bertambahnya elemen senarai (banyak elemen = n), maka bertambah juga space di memori yang diperlukan secara logaritmik. Ini sudah sangat baik, mengingat peningkatan secara logaritmik cenderung lamban. Namun, bubble sort jauh lebih memakan space yang kecil, yaitu O(1) karena operasi yang dilakukan hanyalah swapping dan tidak perlu menggunakan rekursif seperti divide and conquer.

Persoalan yang dihadapi terhadap puzzle 15 ini merupakan persoalan efisiensi. Yaitu mencari tahu seberapa cepat kedua algoritma, lalu membandingkannya dengan test case yang sudah disediakan untuk masing masing algoritma.

Persoalan kedua yang dihadapi adalah mengetes hipotesis bahwa kedua algoritma memiliki hasil langkah langkah yang sama. Bila semua test case benar, maka hipotesis tidak akan ditolak karena kurangnya data, namun bila ada satu saja yang salah, berarti hipotesis ditolak, dan kita dapat mengecek algoritma mana yang memiliki langkah paling sedikit. Namun, perlu diingat bahwa walaupun ada suatu algoritma yang lebih sedikit jumlah langkahnya, kita tidak dapat menyimpulkan algoritma tersebut lebih mangkus karena pendekatan yang dilakukan bersifat statistik(berdasarkan data), bukan scientifik (jelas dan terbukti).

IV. HIPOTESIS

H0 : Algoritma IDA* lebih cepat daripada algoritma Branch and Bound dalam menyelesaikan permainan puzzle 15

H1: Algoritma IDA* tidak lebih cepat daripada algoritma Branch and Bound dalam menyelesaikan permainan puzzle 15

V. TEST CASE

Ada 7 test case yang akan digunakan pada pengujian kali ini. Berikut adalah daftar test case:

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

	1	3	4
5	2	7	8
9	6	10	12
13	14	11	15

5	1	7	3
9	2	11	4
13	6	15	8
	10	14	12

9	2	8	12
11	3	15	10
6		13	5
14	4	1	7

11	12	8	14
10	16	15	7
6	9	3	13
2	4	1	5

2	5	13	12
1		3	15
9	7	14	6
10	11	8	4

11	4	12	2
5	10	3	15
14	1	6	7
	9	8	13

VI. Keterangan Percobaan

Percobaan ini Penulis lakukan menggunakan laptop dengan spesifikasi :

RAM : 8 GB

OS : Windows

Processor: Intel Core i7-7700HQ CPU @ 2.80GHz

Setiap *Test Case* akan dicoba 3 kali dan lama eksekusinya akan didapat dari rata rata 3 percobaan tersebut. Lama eksekusi dalam satuan detik (sekon).

VII. Hasil

Untuk percobaan, lama eksekusi maksimal adalah 1 menit, bila telah melebihi 5 menit, akan dituliskan (-)

Tabel hasil percobaan Branch and Bound:

Nama *Test Case*	Lama Eksekusi	Jumlah Gerakan
Test Case 1	0.0001 sekon	3
Test Case 2	0.003 sekon	6
Test Case 3	0.0057 sekon	15
Test Case 4	-	-
Test Case 5	-	-
Test Case 6	-	-
Test Case 7	-	-

Tabel hasil percobaan IDA*:

Nama *Test Case*	Lama Eksekusi	Jumlah Gerakan
Test Case 1	0 sekon	3
Test Case 2	0.0013 sekon	6
Test Case 3	0.00103 sekon	15
Test Case 4	25.3844 sekon	51
Test Case 5	-	-
Test Case 6	66.4286 sekon	48
Test Case 7	25.2207 sekon	49

VIII. Analisis

Dari data yang diperoleh, terlihat jelas bahwa IDA* untuk puzzle15 lebih cepat daripada Branch and Bound. Selain lebih cepat, dari teori sendiri, IDA* jauh lebih ringan dari memori, karena sifatnya yang merupakan deep first search sehingga tidak harus menyimpan "history" dari setiap node yang dikunjungi.

Pada saat melakukan percobaan, ada beberapa hal yang terjadi, terutama pada lama eksekusi yang diberi tanda (-). Pada pengujian *Test Case* 5 IDA*, sebenarnya telah keluar hasil yang menyatakan jumlah gerakan 62, namun karena lama eksekusinya yaitu sekitar 6 menit, maka tidak Penulis masukan ke dalam laporan, namun ini menunjukan bahwa IDA* dapat menyelesaikan puzzle bahkan dengan jumlah gerakan yang diperlukan 62.

Di sisi lain, algoritma Branch and Bound saat mengeksekusi *Test Case* 4 sampai 7 (*Test Case* dengan banyak gerakan diperlukan) tidak bisa mendapatkan goal node. Di *Test Case* 4, Penulis menunggu lebih dari 5 menit, lalu mendapat pesan bahwa terjadi memory error. Ini menunjukan bahwa memori yang dipakai algoritma Branch and Bound cukup besar sampai menimbulkan memory error, mengingat laptop Penulis memiliki ram laptop yang Penulis gunakan 8 GB.

IX. Kesimpulan

Pernyataan H0, "algoritma IDA* lebih cepat daripada algoritma Branch and Bound untuk permainan puzzle 15", tidak ditolak karena kurangnya data untuk mendukung pernyataan H1, "algoritma IDA* tidak lebih cepat daripada algoritma Branch and Bound dalam permainan puzzle 15"

X. Video Link Youtube

https://www.youtube.com/watch?v=CbJubzkBYiU&t=1s – Penjelasan algoritma walking distance

XI. Penutup

Pertama, penulis mengucapkan syukur kepada Tuhan Yang Maha Esa karena atas rahmat-Nya, penulis bisa menyelesaikan makalah ini dengan baik. Setelah itu, penulis juga ingin mengucapkan terima kasih kepada Ibu Fariska atas bimbingannya dan dukungannya dalam pengajaran mata kuliah IF2211 Strategi Algoritma. Penulis juga ingin mengucapkan terima kasih kepada teman teman penulis yang telah memberikan inspirasi dan semangat

kepada penulis dalam menulis laporan ini.

Penulis juga menyadari bahwa masih ada banyak kekurangan dalam laporan ini dan penulis meminta maaf apabila ada kekurangan dalam penamaan, penulisan, struktur dari penyusunan dokumen ini. Segala saran yang membangun akan sangat penulis apresiasi. Semoga makalah ini dapat bermanfaat dan bisa dijadikan bahan referensi atau bacaan lebih lanjut kedepannya.

XII. REFERENSI

[1] Munir, Rinaldi. 2018. *Slide A-Star-Best-FS-dan-UCS(2018)*. Bandung, Sekolah Tinggi Elektro dan Informatika Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/stima19-20.htm

[2] Munir, Rinaldi. 2018. *Slide Algoritma-Branch-&-Bound-(2018)*. Bandung, Sekolah Tinggi Elektro dan Informatika Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/stima19-20.htm

[3] User "algorithmsinsight". 2016. IDA-Star(IDA*) Algorithm in General. Diakses 23 April 2020 pukul 16.00. sumber : https://algorithmsinsight.wordpress.com/graph-theory-2/ida-star-algorithm-in-general/

[4] Channel "John Levine". 2018. *Iterative Deepening*. Diakses 24 April, 2020 pukul 17.00. sumber: *http://rutepesawat.blogspot.com/2011/11/rute-penerbangan-lion-air.html*.

[5] User "orlp". 2018. *PyPy, 195 moves, ~12 seconds computation.* Diakses 24 April 2020 pukul 12.00. sumber: https://codegolf.stackexchange.com/questions/6884/solve-the-15-puzzle-the-tile-sliding-puzzle

[6] Garg, Prateek. 2016. *Breadth First Search*. sumber : https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

[7] Takahashi, Ken'ichiro. 2015. *パズル自動解答プログラムの作り方(Cara membuat jawaban teka teki otomatis)*. Diakses 24 April 2020 pukul 14.00. sumber : http://www.ic-net.or.jp/home/takaken/nt/slide/solve15.html

PERNYATAAN

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

Bogor, 24 April 2020

Mario Gunawan / 13518114