Perbandingan Waktu yang Dibutuhkan Untuk Menyelesaikan 15-Puzzle dengan Menggunakan Heuristik Fungsi Kurang dan Manhattan Distance

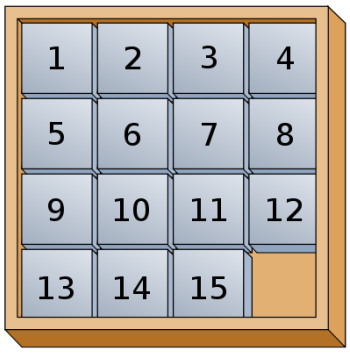
Stefanus Gusega Gunawan – 13518149
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): gunawanstefanus006@gmail.com

***Abstrak*—*15-Puzzle* adalah sebuah permainan sederhana yang bertujuan untuk menyusun 15 angka menjadi satu pola tertentu. Selain menggunakan nalar manusia, permainan ini bisa diselesaikan dengan menggunakan berbagai strategi algoritma dalam pemrograman. Kali ini, akan dicoba membandingkan manakah yang paling cepat dalam menyelesaikan *15-Puzzle* ini. Algoritma yang akan dicoba adalah A* yang merupakan spesialisasi *Branch and Bound* dengan menggunakan dua heuristik yang berbeda, yaitu heuristik penjumlahan *Manhattan Distance* dan penjumlahan fungsi kurang. Dalam makalah ini, akan dijelaskan bagaimana perbandingan waktu yang dibutuhkan untuk menyelesaikan beberapa *start state*, dan dimodelkan secara sederhana menggunakan persamaan regresi linier.**

Kata Kunci*—*algoritma, waktu, permainan, perbandingan, A*, Branch and Bound, heuristik

I. PENDAHULUAN

15-Puzzle adalah permainan sederhana yang sudah popular selama lebih dari seratus tahun. Permainan ini terlihat cukup sederhana, bahkan bisa dimainkan oleh anak kecil. Untuk orang-orang yang jarang bermain *puzzle* – bahkan orang dewasa sekalipun – akan memakan waktu yang lama dalam menyelesaikan. Mungkin, jika mereka berfokus untuk menyelesaikan, bisa memakan waktu sekitar 10 hingga 15 menit.



Gambar 1 : Contoh papan *15-puzzle*

Sumber : guptaanna.github.io

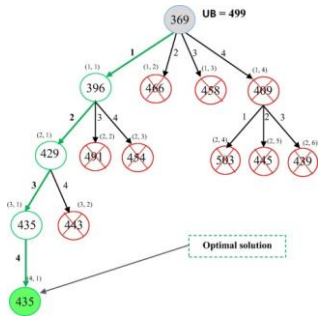
State awal dari permainan *15-Puzzle* ini adalah *15-Puzzle* yang sudah diacak semua angkanya, sehingga setidaknya ada satu angka yang tidak pada posisinya, dan *goal state*-nya adalah menjadikan *puzzle* tersebut berada pada posisi yang *solved*[1]. Posisi *solved* ini bisa bermacam-macam tergantung pembuat permainannya. Pada makalah ini, akan digunakan *solved position* dengan posisi seperti pada Gambar 1.

Permainan *puzzle* ini dapat diselesaikan dengan menggunakan algoritma A*, dengan menggunakan fungsi heuristik yang mampu memprediksi sedekat mungkin berapa jumlah langkah yang mungkin untuk mencapai *goal state*[1]. Lalu, akan ditentukan algoritma manakah yang lebih cepat dalam menyelesaikan *puzzle* ini. Perlu diperhatikan, heuristic yang bisa diterapkan sangat banyak, dan pada makalah ini akan dijelaskan dengan menggunakan heuristik fungsi kurang dan *Manhattan distance*.

II. DASAR TEORI

A. *Algoritma Branch and Bound*

Algoritma *Branch and Bound* adalah sebuah paradigma desain algoritma yang digunakan untuk menyelesaikan permasalahan yang memiliki kompleksitas hingga eksponensial dan faktorial, yang memungkinkan untuk menelusuri semua kemungkinan yang ada (*exhaustive search*). Algoritma ini dimodelkan dalam bentuk *state space tree*, atau yang biasa disebut dengan pohon ruang status, di mana di tiap *node* terdapat *cost*-nya masing-masing. Seperti yang telah dikatakan sebelumnya, algoritma *Branch and Bound* tidak akan menelusuri semua kemungkinan yang ada.



Gambar 2 : Penggambaran pohon beserta *cost*

Sumber : ScienceDirect.com

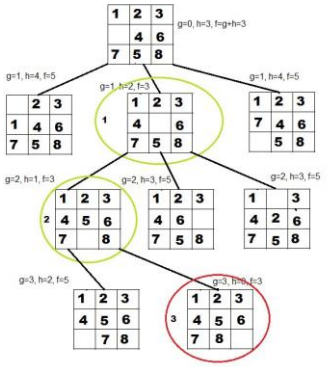
Algoritma *Branch and Bound* melakukan pencarian pada seluruh ruang yang merupakan kandidat solusi, dengan menggunakan suatu trik yang sangat mengefisiensikan waktu[2]. Trik tersebut adalah mengatur prioritas mana dulu yang ditelusuri dari tiap *node* yang dibangkitkan dengan menggunakan estimasi *cost* pada tiap *node*[2]. Berdasarkan pada estimasi *cost*, lalu *node-node* yang hidup disusun ke dalam suatu *priority queue*, supaya tahu mana dulu yang harus dibangkitkan dan ditelusuri.

Misal, *g(x)* adalah perkiraan dari usaha atau jumlah langkah yang akan dilalui untuk mencapai *answer node* dari *node x*[3]. *Node* x dimasukkan ke dalam fungsi *c(x)*, sehingga *c(x) = f(h(x))+g(x)*, dengan *h(x)* adalah *cost* untuk mencapai *node x* dari *root node*[3]. Akan tetapi, *cost* akan disederhanakan menjadi *c(x) = h(x)+g(x)*. Lalu, kita akan memilih *node* dengan *cost* terkecil, untuk di-*expand*, untuk menghasilkan suatu solusi dengan *cost* yang minimal.

B. Algoritma A (A-star)*

Algoritma A* (*A-star*) adalah salah satu algoritma pencarian yang cukup popular di kalangan *programmer* dalam mencari solusi yang optimal[4]. Optimal yang dimaksud di sini adalah algoritma ini dapat mencari solusi yang membutuhkan *cost* terkecil atau *path* terpendek untuk mencapai solusi tersebut. Algoritma A* adalah algoritma yang berbasis BFS (*Breadth-First Search*) yang sudah sangat *advanced*[4]. Mengapa? Karena algoritma ini juga sudah mempertimbangkan berapa kira-kira *cost* yang akan dihabiskan saat mengekspan sebuah simpul.

Algoritma A* adalah algoritma yang optimal dan *complete*[4]. Algoritma yang optimal, karena algoritma ini pasti menemukan *cost* terkecil yang dibutuhkan dari simpul awal ke simpul solusi[4]. Algoritma yang *complete* atau lengkap, karena algoritma ini akan menemukan semua solusi yang optimal[4]. Sehingga, hal ini membuat algoritma A* adalah algoritma yang terbaik di sebagian kasus pencarian solusi.



Gambar 3 : Contoh pohon ruang status untuk *8-puzzle* menggunakan A*

Sumber : blog.goodaudience.com

*C. Hubungan antara Algoritma Branch and Bound dan A**

Jika dilihat sekilas, kedua algoritma ini terlihat mirip. Kedua algoritma ini sama-sama menghitung perkiraan dari *node* tertentu ke *solution node*. Dan perhitungan *cost*-nya pun juga sama persis, yaitu *cost = root node to certain node + certain node to answer node*. Mengapa bisa terjadi demikian?

Branch and Bound adalah suatu algoritma atau teknik untuk memecahkan masalah yang sudah digunakan pada berbagai masalah dan persoalan yang ditemui pada *operation research* dan *combinatorial mathematics*[5]. Seperti yang sudah dijelaskan, *branch and bound* meng-*expand* simpul yang memiliki *cost* yang terkecil, sehingga bisa mengefisiensi waktu untuk tidak menelusuri semua pola yang tidak menuju ke solusi yang optimal. *Branch and bound* pun juga sama-sama menggunakan fungsi heuristik agar kedua algoritma itu bisa menentukan *cost* dari masing-masing simpul.

Menurut salah satu jurnal yang ditemukan di internet, dikatakan bahwa sebenarnya ada hubungan antara *branch and bound* dan A*[5]. Sang penulis jurnal juga mengakui, jikalau beberapa heuristik pada *artificial intelligence* sangat berhubungan dengan prosedur dari *branch and bound*[5]. Di dalam jurnal itu, ditunjukkan dua algoritma *artifial intelligence*, yaitu algoritma A* dan AO* adalah kasus special atau bisa dibilang spesialisasi dari formulasi atau perumusan umum *branch and bound*[5].

D. Heuristik

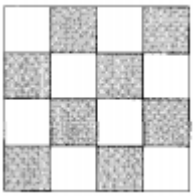
Untuk menentukan *cost* dari tiap simpul yang sudah dibangkitkan, diperlukan bantuan dari fungsi heuristik untuk membantu membuat estimasi, kira-kira simpul manakah yang lebih dekat menuju ke solusi. Di dalam makalah ini, akan dibandingkan dua macam heuristik. Kedua heuristik itu adalah fungsi kurang dan *Manhattan distance*.

Fungsi kurang adalah salah satu heuristik yang digunakan untuk menyelesaikan permainan *15-puzzle.* Cara kerja fungsi heuristik ini adalah sebagai berikut: untuk setiap *state*, misal *kurang(i)* adalah jumlah dari *tile j* yang menyebabkan *j < i* dan *position(j) > position(i)*[4].

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

Gambar 4 : Contoh penyusunan *state* awal dari *15-puzzle*
Sumber : Ellis Horowitz, *Fundamentals of Computer Algorithms*, (New York: Computer Science Press, 1997), 383. Buku Elektronik.

Sebagai contoh pada Gambar 4, *kurang(1) = 0*, karena tidak ada angka yang lebih besar dari 1 pada nomor *tile* setelah 1, *kurang(4) = 1*, karena terdapat angka 2 yang lebih kecil dari 4, *kurang(12) = 6*, karena ada angka 7, 6, 11, 8, 9, dan 10 yang lebih kecil dari 12, dan seterusnya[4]. Dengan penggunaan *kurang(i)*, juga bisa ditentukan apakah *state puzzle* ini dapat diselesaikan atau tidak.

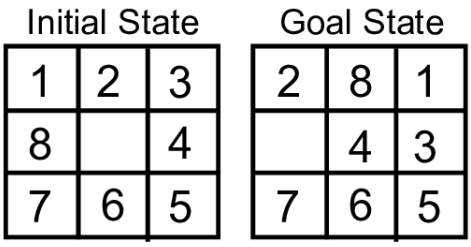


Gambar 5 : Bantuan untuk mengetahui apakah *state puzzle* dapat diselesaikan (*reachable*)
Sumber : Ellis Horowitz, *Fundamentals of Computer Algorithms*, (New York: Computer Science Press, 1997), 383. Buku Elektronik.

Misalkan x = 1 jika letak *tile* yang kosong terletak pada daerah yang diarsir, dan x = 0, jika sebaliknya[4]. Sebuah *state* dari suatu *puzzle* dapat diselesaikan atau *reachable*, jika dan hanya jika jumlah dari *kurang*(*i*) dari *i = 1* sampai *i = 16*, lalu ditambah dengan x bernilai genap[4]. Dan, pada akhirnya heuristik yang digunakan adalah jumlah dari *kurang*(*i*) dari *i = 1* sampai *i = 16*[4].

Heuristik kedua adalah dengan memanfaatkan penjumlaha dari seluruh *Manhattan Distance*. *Manhattan Distance* adalah jarak absolut secara horizontal ditambah jarak absolut secara vertikal dari setiap *tile* ke lokasinya yang benar[1]. Sudah dipastikan bahwa heuristik *Manhattan Distance* ini *admissible*[1]. Penyelesaian *15-puzzle* dengan menggunakan algoritma A* dengan heuristik ini benar-benar sangat memotong waktu penyelesaian, yang artinya waktu untuk menyelesaikannya sangat cepat[1]. Maka dari itu, heuristik ini menarik untuk dibandingkan dengan penjumlahan dari heuristik fungsi kurang.

Cara kerja heuristik *Manhattan Distance* ini adalah sebagai berikut.



Gambar 6 : Contoh *initial state* dan *goal state* dari sebuah *8-puzzle*
Sumber : goodaudience.com

Kita ambil contoh dari Gambar 6, yaitu sebuah *initial state* dan *goal state* dari sebuah *8-puzzle*. *8-puzzle* adalah variasi dari *15-puzzle*, hanya dengan jumlah *tile* yang lebih sedikit. Kita lihat *tile* dengan nomor 1. *Manhattan Distance* dari *tile* bernomor 1 tersebut adalah 2. Mengapa? Karena pada *initial state*, *tile* bernomor 1 itu ada pada baris pertama dan kolom pertama. Sedangkan, pada *goal state*, *tile* bernomor 1 berada pada baris pertama dan kolom ketiga. Sehingga, *Manhattan Distance* adalah |1-1|+|3-1| = 0+2 = 2. Ambil contoh lagi, *tile* bernomor 8. Pada *initial state*, *tile* bernomor 8 terletak pada baris kedua dan kolom pertama. Namun, pada *goal state* terletak pada baris pertama dan kolom kedua. Sehingga, *Manhattan Distance* adalah |1-2|+|2-1| = 1+1 = 2. Begitu juga, untuk *tile* yang lainnya.

III. Perancangan Algoritma

A. *Paradigma pemrograman yang digunakan*

Pada percobaan untuk kepentingan makalah ini, akan digunakan salah satu paradigma pemrograman yang cukup populer dalam suatu pembangunan aplikasi, yaitu paradigm Pemrograman Berorientasi Objek atau *Object-Oriented Programming*. Di dalam paradigma tersebut, dikenal ada istilah objek, kelas, pewarisan sifat – atribut – (*inheritance*), *polymorphism*, dan sebagainya[6]. Namun, pada percobaan ini, tidak akan menggunakan sifat-sifat yang berhubungan dengan pewarisan sifat seperti *inheritance* dan *polymorphism*, guna mempercepat waktu perancangan algoritma.

Dalam percobaan ini, hanya akan digunakan konsep pemrograman berorientasi objek yang mengutamakan objek *15-puzzle* ini memiliki atributnya masing-masing dalam menyelesaikan persoalan untuk setiap heuristik atau metode yang digunakan. *15-puzzle* sendiri akan dikemas dalam bentuk matriks berukuran empat kali empat.

B. *Atribut-atribut pada Penyelesaian Puzzle*

Karena paradigma pemrograman berorientasi objek, maka akan dimanfaatkan atribut-atribut dari objek *puzzle* dengan heuristik penjumlahan fungsi kurang. Berikut adalah daftar atribut yang akan digunakan:

- *Matrix*, guna menyimpan pada *state* tersebut, susunan *puzzle* yang direpresentasikan dalam bentuk *matrix* itu seperti apa.
- *Cost*, guna menyimpan berapa *cost* untuk mencapai *state* tersebut.
- *BlankX*, guna menyimpan pada baris keberapakah, *tile* yang kosong sekarang.
- *BlankY*, guna menyimpan pada kolom keberapakah, *tile* yang kosong sekarang.
- *Solvable*, atribut bertipe *Boolean*, guna menyimpan apakah *puzzle* pada *state* tersebut dapat diselesaikan atau tidak.
- *Path*, atribut bertipe *array of Puzzle*, guna menyimpan *solution path*, dari *state* awal hingga ke *goal state*.

- *Step*, atribut bertipe *integer* yang digunakan untuk menyimpan sudah berapa *step*-kah dari *state* awal hingga *state* ini.
- *GeneratedNode*, atribut yang bertipe *integer* yang digunakan untuk menyimpan sudah berapa simpul yang dibangkitkan ketika mencapai *state* tersebut.
- *History*, atribut yang bertipe *map* dengan *key* bertipe *array* yang dibentuk dari *matrix* yang sudah di-*flatten* dan *value* bertipe *Boolean*. Mengapa menggunakan *map*? Menurut salah satu sumber yang digunakan, dibandingkan menggunakan *array* untuk menyimpan *history*, lebih baik menggunakan *set/map*, karena kompleksitas yang digunakan mendekati O(1)[7]. Sudah jelas, bahwa jauh lebih efisien untuk data yang berukuran besar[7].

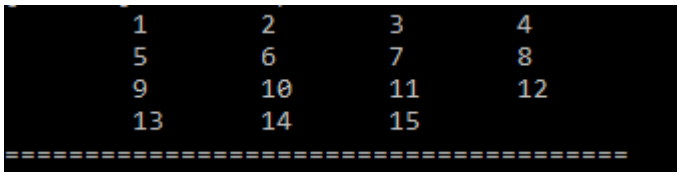
C. *Method-method yang digunakan untuk menyelesaikan Puzzle*

Berikut ini *method-method* yang digunakan untuk menyelesaikan persoalan *15-puzzle*:

- *Constructor* dengan menggunakan *filename*: nama *file*. Di dalam *constructor*, matriks yang sudah ada pada *file* dengan *extension* .txt, akan dimasukkan ke matriks pada program dengan tipe *integer*. Di dalam sini, *cost state* awal juga dihitung. Dan, semua atribut-atribut diinisialisasi.
- *Append*, guna menambahkan *path* dari *puzzle* ke dalam atribut *path.*
- *kurangTambahX*, sebagai *method* bantuan untuk menentukan apakah *puzzle* dapat diselesaikan atau tidak.
- *isSolvable*, mengembalikan nilai *Boolean* sebagai penentu apakah *puzzle* dapat diselesaikan atau tidak.
- *Up*, membuat *blank tile* bergerak ke atas jika mungkin dan menambahkannya ke *path* agar disimpan terlebih dahulu, *generatedNode* bertambah satu, *step* dan *cost* bertambah satu.
- *Down*, membuat *blank tile* bergerak ke bawah jika mungkin dan menambahkannya ke *path* agar disimpan terlebih dahulu, *generatedNode* bertambah satu, *step* dan *cost* bertambah satu.
- *Left*, membuat *blank tile* bergerak ke kiri jika mungkin dan menambahkannya ke *path* agar disimpan terlebih dahulu, *generatedNode* bertambah satu, *step* dan *cost* bertambah satu.
- *Right*, membuat *blank tile* bergerak ke kanan jika mungkin dan menambahkannya ke *path* agar disimpan terlebih dahulu, *generatedNode* bertambah satu, *step* dan *cost* bertambah satu.
- *View*, untuk menampilkan matriks
- *Operator overloading (<)*, untuk membandingkan mana dulu yang lebih prioritas, yaitu dengan *cost*-nya yang dibandingkan.
- *Copy*, digunakan untuk meng-*copy* semua atribut yang ada, dan mengembalikan dalam tipe *Puzzle*.
- *Flatten matrix*, digunakan untuk membuat *matrix* menjadi *array* satu dimensi.
- *Inversion*, merupakan penjumlahan dari fungsi kurang.
- *Manhattan*, merupakan penjumlahan dari *manhattan distance*.
- *printPath*, merupakan prosedur untuk mencetak *path* yang sudah ditemukan. *Path* dicetak dalam bentuk urutan pergerakan matriks.
- *Solve*, prosedur yang digunakan untuk *solving the puzzle*.

D. *Goal state dan initial state*

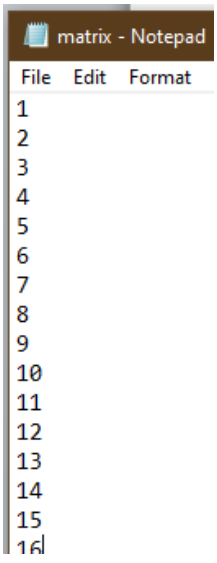
Goal state yang digunakan pada persoalan *15-puzzle* kali ini adalah dengan susunan sebagai berikut.



Gambar 7 : *Goal state* yang akan digunakan untuk percobaan pada makalah ini

Sumber : Dokumentasi pribadi

Dan berikut *format* untuk *state* awal pada *file* dengan *extension* .txt.

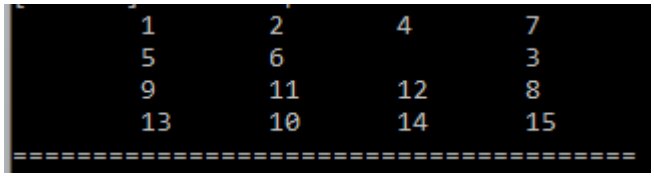


Gambar 8 : Contoh konfigurasi untuk *state* awal pada *file* txt.
Sumber : Dokumentasi pribadi

E. Pengujian

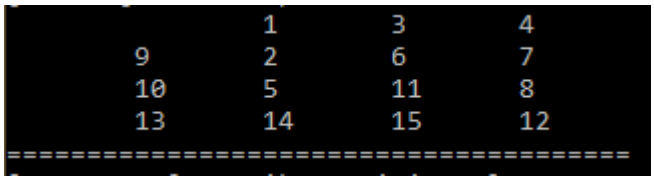
Pengujian dilakukan dengan menyelesaikan tiga persoalan *15-puzzle* yang berbeda, dan waktu dalam menyelesaikannya akan disimpan ke dalam satuan detik. Pengujian akan dilakukan sebanyak lima kali.

Pengujian pertama akan dicoba persoalan seperti gambar di bawah ini.



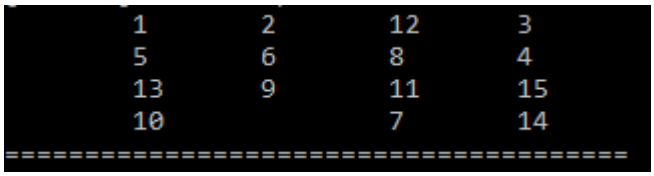
Gambar 9 : Persoalan pertama yang akan dicoba
Sumber : Dokumentasi Pribadi

Lalu, pengujian kedua akan dicoba persoalan seperti pada gambar berikut.



Gambar 10 : Persoalan kedua yang akan dicoba
Sumber : Dokumentasi pribadi

Lalu, pada pengujian ketiga akan dicoba persoalan seperti pada gambar berikut.



Gambar 11 : Persoalan ketiga yang akan dicoba
Sumber : Dokumentasi pribadi

Lalu, akan dicoba persoalan pertama terlebih dahulu sebanyak lima kali dan akan direkam atau disimpan waktu yang dihabiskan untuk menyelesaikan. Berikut adalah hasil dari percobaan yang dilakukan disajikan dalam bentuk tabel.

Percobaan ke-	**Waktu yang dibutuhkan jika menggunakan heuristik penjumlahan fungsi kurang**	**Waktu yang dibutuhkan jika menggunakan heuristik penjumlahan *Manhattan Distance***
1	0.3219935894012451 detik	0.30900049209594727 detik
2	0.38800787925720215 detik	0.34599757194519043 detik
3	0.31704211235046387 detik	0.3130185604095459 detik
4	0.3220028877258301 detik	0.3850083351135254 detik
5	0.3160114288330078 detik	0.31111836433410645 detik
Rata-rata	0.3330115795135498 detik	0.3328286647796631 detik

Tabel 1 : Hasil percobaan dengan persoalan pertama
Sumber : Dokumentasi pribadi

Lalu, dilanjutkan ke pengujian pada persoalan kedua sebanyak lima kali pula, lalu disimpan ke dalam tabel berikut ini.

Percobaan ke-	**Waktu yang dibutuhkan jika menggunakan heuristik penjumlahan fungsi kurang**	**Waktu yang dibutuhkan jika menggunakan heuristik penjumlahan *Manhattan Distance***
1	0.15004324913024902 detik	0.14999842643737793 detik
2	0.1510453224182129 detik	0.1459941864013672 detik
3	0.19100093841552734 detik	0.1869974136352539 detik
4	0.17899441719055176 detik	0.1530148983001709 detik
5	0.2010021209716797 detik	0.19300484657287598 detik
Rata-rata	0.17441720962524415 detik	0.1658019542694092 detik

Tabel 2 : Hasil percobaan dengan persoalan kedua
Sumber : Dokumentasi pribadi

Lalu, akan diuji persoalan ketiga sebanyak lima kali pula dan akan direkam dan disimpan waktu yang digunakan untuk menyelesaikan. Berikut adalah hasil dari percobaan yang dilakukan disajikan dalam bentuk tabel.

Percobaan ke-	**Waktu yang dibutuhkan jika menggunakan heuristik penjumlahan fungsi kurang**	**Waktu yang dibutuhkan jika menggunakan heuristik penjumlahan *Manhattan Distance***
1	86.80696201324463 detik	94.08887124061584 detik
2	90.85305762290955 detik	92.3167462348938 detik
3	90.63323950767517 detik	95.89204239845276 detik

4	94.6835389137268 detik	95.28112077713013 detik
5	92.67506551742554 detik	96.92058491706848 detik
Rata-rata	91.13037271499634 detik	94.8998731136322 detik

Tabel 3 : Hasil percobaan dengan persoalan ketiga
Sumber : Dokumentasi pribadi

Dari data-data yang sudah didapat dan direkam, didapat bahwa setelah diberikan lima kali percobaan untuk setiap satu persoalan, jumlah simpul yang dibangkitkan pun juga sama dan jumlah *step* yang dilalui pun sama. Maka dari itu, hanya bisa dibandingkan lewat waktu penyelesaian saja.

Jika dilihat dari waktu dalam menyelesaikan, terlihat bahwa penggunaan heuristik *Manhattan Distance* hanya optimal dalam konteks waktu hanya pada saat jumlah *step*-nya kecil. Jika dimodelkan ke dalam grafik regresi linier, akan dihasilkan persamaan regresi dengan pemodelan sebagai berikut (y menandakan waktu yang dibutuhkan dan x menunjukkan jumlah langkah yang dilalui).

Persamaan regresi yang dihasilkan dari penggunaan heuristik fungsi kurang	**Persamaan regresi yang dihasilkan dari penggunaan heuristik *Manhattan Distance***
Y = 9.488*X - 99.13	Y = 9.882*X - 103.3

Tabel 4 : Hasil perhitungan persamaan regresi linier dari masing-masing penggunaan heuristik
Sumber : Dokumentasi Pribadi

Dilihat dari gradien persamaan regresi linier yang dihasilkan, terlihat bahwa penggunaan heuristik *Manhattan Distance* memakan waktu sedikit lebih lama dalam menyelesaikan *15-puzzle* ini.

Namun, jika dilihat dari berbagai sumber yang ada di internet, heuristik *Manhattan Distance* adalah heuristik yang sering digunakan dalam penyelesaian *15-puzzle* dan dipastikan salah satu yang tercepat[8]. Mungkin, dalam percobaan yang dilakukan ini tidak memperhatikan berbagai faktor pada kode program. Hipotesis saya, ini bisa terjadi karena saya hanya mengubah penggunaan heuristiknya saja. Dan, tidak menghapus kode-kode yang sekiranya tidak penting dalam penyelesaian *15-puzzle*. Hingga saat ini, saya masih belum menemukan bagian manakah yang membuat *Manhattan* terlihat lambat.

IV. Kesimpulan

Jika dilihat dari percobaan yang telah dilakukan, dengan mengabaikan beberapa faktor, maka bisa disimpulkan bahwa penggunaan heuristik penjumlahan *Manhattan Distance* memakan waktu yang sedikit lebih lama daripada heuristik penjumlahan fungsi kurang. Hal ini dikarenakan, kode program yang berbeda hanya pada letak heuristiknya, sedangkan kode-kode program yang tidak penting tidaklah dihapus, maka dari itu bisa memakan waktu yang lama.

V. Pranala Video Di Youtube

Berikut *link* video di YouTube: https://youtu.be/5wSUCmvVVns.

VI. Ucapan Terimakasih

Puji syukur saya ucapkan kepada Tuhan Yang Maha Esa, dengan penyertaan-Nya senantiasa, saya dapat menyusun sebuah makalah yang merupakan percobaan kecil-kecilan saya untuk memenuhi Tugas Makalah Strategi Algoritma Tahun Ajaran 2019/2020. Terima kasih juga saya ucapkan kepada kedua orangtua saya, yang senantiasa selalu mendukung saya, baik secara moral maupun moriil. Terimakasih juga saya ucapkan kepada tim dosen Strategi Algoritma, yang sudah senantiasa membimbing kami dalam mengerti berbagai strategi-strategi dasar algoritma pada pemrograman selama satu semester. Tanpa bantuan bapak ibu sekalian, saya tidak dapat menyelesaikan makalah ini.

Referensi

[1] Jensen, Presten (2017, 31 Oktober). *Solving the 15-Puzzle*. Dikutip 30 April 2020 dari Medium: https://medium.com/breathe-publication/solving-the-15-puzzle-e7e60a3d9782.

[2] Lipton, R. J. (2012, 19 Desember). *Branch And Bound—Why Does It Work?*. Dikutip 1 Mei 2020 dari Wordpress: https://rjlipton.wordpress.com/2012/12/19/branch-and-bound-why-does-it-work/.

[3] Horowitz, Ellis; Sartaj Sahni; Sanguthevar Rajasekaran. 1997. *Computer Algorithms*. New York: Computer Science Press.

[4] Akash (2019, 4 Desember). *What is the A* Algorithm and How does it work?*. Dikutip 1 Mei 2020 dari Edureka: https://www.edureka.co/blog/a-search-algorithm/.

[5] Nau, Dana S.; Vipin Kumar; Laveen Kanal. 1984. *General Branch and Bound, and Its Relation to A* and AO*. *Artificial Intelligence*. 23(1): 29-58.

[6] Petkov, Alexander (2018, 27 Juni). *How to explain object-oriented programming concepts to a 6-year-old*. Dikutip 3 Mei 2020 dari FreeCodeCamp: https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/.

[7] Babu, Rajesh (2018, 30 April). *Array vs Set vs Map vs Object – Real-time use cases in JavaScript (ES6/ES7)*.Dikutip 3 Mei 2020 dari CodeBurst: https://codeburst.io/array-vs-set-vs-map-vs-object-real-time-use-cases-in-javascript-es6-47ee3295329b.

[8] Korf, R. E. 2000. *Recent Progress in the Design and Analysis of Admissible Heuristic Functions*. Makalah. Dalam Choueiry, B. Y.

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.

Surabaya, 3 Mei 2020

Stefanus Gusega Gunawan
13518149