Pemanfaatan Algoritma *A** dalam Penyelesaian 15 *Puzzle Game*

Evan Pradanika - 13518126
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518126@std.stei.itb.ac.id

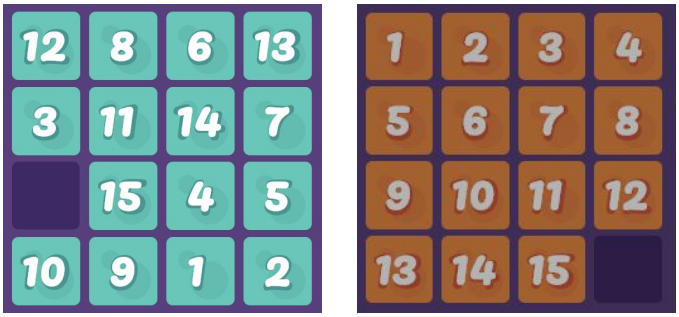
***Abstract*—Algoritma *A** adalah algoritma komputasi yang digunakan untuk mencari solusi secara optimal. Algoritma ini bisa digunakan untuk mencari solusi dari banyak hal, salah satunya yaitu permainan *sliding puzzle* seperti 15 *Puzzle*. 15 *Puzzle* sendiri merupakan sebuah *puzzle* berukuran 4 x 4 yang berisikan angka 1-15 dan sebuah ubin kosong. Solusi dari *puzzle* ini yaitu menghasilkan *puzzle* yang sudah berurut dari angka 1 hingga 15 dengan ubin kosong berada di ubin terakhir.**

Keywords—A*, optimal, 15 Puzzle, berurut

I. Pendahuluan

15 *Puzzle* merupakan salah satu contoh dari permainan *sliding puzzle* yang pertama kali ditemukan pada Januari 1880 di Amerika Serikat. Penemu permainan ini adalah Noyes Palmer Chapman, seorang *postmaster* dari Canastota, New York. Akan tetapi, *puzzle* ini lebih dikenal diciptakan oleh Sam Loyd, pemain catur yang berasal dari Amerika Serikat.

Ada banyak algoritma yang bisa digunakan untuk menyelesaikan *puzzle* ini, seperti algoritma *Brute Force*, *Depth-First Search*, *Breadth-First Search*, *Backtracking*, *Branch and Bound*, dan *A**. Dari algoritma-algoritma tersebut, tidak semua algoritma dapat menyelesaikan permasalahan 15 *Puzzle* yang kompleks. Umumnya hal ini disebabkan oleh waktu yang terlalu lama yang dibutuhkan oleh algoritma tersebut. Untuk menghindari waktu yang terlalu lama, penulis menggunakan algoritma *A** untuk mencari solusi yang optimal dari permasalah 15 *puzzle*.

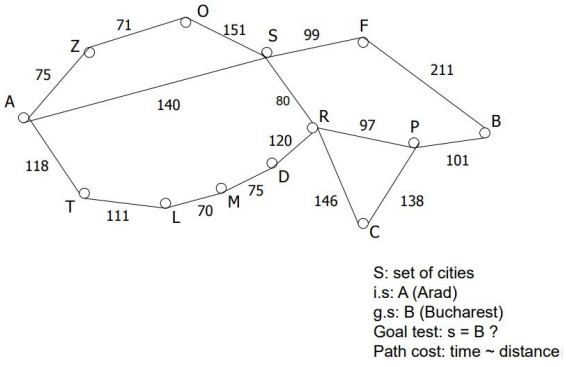


Gambar 1. 15 *Puzzle Game*
Sumber: https://15puzzle.netlify.app/

II. Dasar Teori

A. Path Planning

Path Planning dari lokasi A ke B atau bereaksi terhadap suatu hal merupakan tugas yang mudah untuk manusia. Tetapi untuk suatu program tentunya tidak. Program membutuhkan algoritma penentu dan perencanaan agar bisa menghasilkan hasil yang diinginkan. *Path Planning* sendiri digunakan untuk menyelesaikan banyak permasalahan, dimulai dari mencari rute yang simpel hingga memilih langkah-langkah yang sesuai untuk mencapai suatu tujuan.



Gambar 2. *Path Planning*
Sumber: *Slide* Kuliah Problem Solving and Search, Rinaldi Munir *Halaman* 4

Dalam ilmu komputasi, *Path Planning* merupakan salah satu permasalahan yang bisa diselesaikan dengan algoritma pencarian. Algoritma pencarian sendiri merupakan algoritma yang paling umum digunakan untuk menyelesaikan permasalahan dalam kecerdasan buatan. Di dalam permasalahan kecerdasan buatan, urutan langkah-langkah yang dibutuhkan untuk mencapai solusi merupakan hal yang sangat penting. Secara umum, algoritma pencarian dibagi menjadi dua, yaitu *Uninformed Search Algorithm* dan *Informed Search Algorithm*.

1) Uninformed Search Algorithm

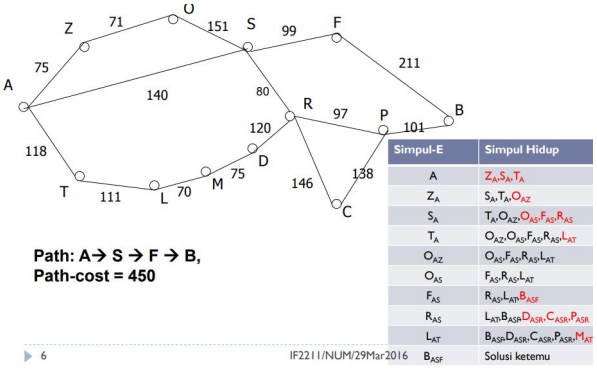
Uninformed Search merupakan salah satu tipe algoritma pencarian yang biasa disebut *Blind Search*. Hal ini disebabkan oleh tidak adanya informasi tambahan tentang kondisi eksternal yang disediakan oleh masalah dalam teknik ini. Singkatnya, *Uninformed Search* adalah algoritma yang tidak memberikan informasi tentang permasalahan yang ada,

melainkan hanya sebatas definisi dari algoritma tersebut. Algoritma ini akan membedakan *state* tujuan dengan *state* non-tujuan. Pencariannya dilakukan berdasarkan urutan yang diinginkan.

Uninformed Search Algorithm sendiri memiliki banyak jenis, seperti:

a) *Breadth-First Search*

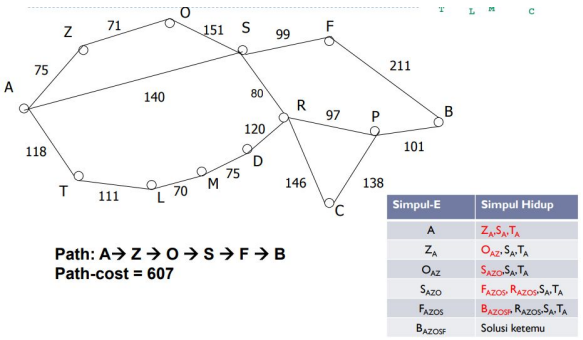
Breadth-First Search merupakan algoritma pencarian yang memprioritaskan dengan kedalaman paling kecil. Algoritma ini akan memeriksa semua node pada kedalaman 1 terlebih dahulu, baru melanjutkan ke simpul-simpul pada kedalaman yang selanjutnya. Hal ini terus dilakukan hingga algoritma menemukan solusi yang diinginkan atau semua simpul yang diperiksa merupakan daun. Algoritma ini memerlukan waktu eksekusi yang tidak begitu lama, tetapi akan memakan memori yang sangat besar dikarenakan banyaknya simpul yang dibangkitkan.



Gambar 3. *Breadth-First Search*
Sumber: *Slide* Kuliah Problem Solving and Search, Rinaldi Munir
Halaman 6

b) *Depth-First Search*

Depth-First Search merupakan algoritma pencarian yang memprioritaskan dengan kedalaman paling besar. Algoritma ini akan memeriksa hanya 1 node pada kedalaman 1 dan melanjutkan ke salah satu simpul yang telah dibangkutkan pada kedalaman yang selanjutnya. Hal ini terus dilakukan hingga algoritma menemukan solusi yang diinginkan atau impul yang diperiksa merupakan daun. Algoritma ini tentunya tidak membutuhkan memori yang sangat besar dikarenakan simpul yang dibangkitkan lebih sedikit. Akan tetapi, algoritma ini memerlukan waktu eksekusi yang sangat lama dikarenakan hanya mencari pada 1 akar hingga daun.



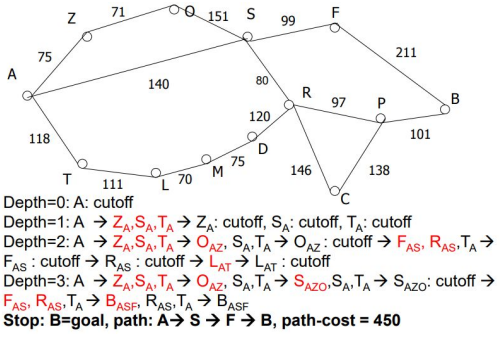
Gambar 4.*Depth-First Search*
Sumber: *Slide* Kuliah Problem Solving and Search, Rinaldi Munir
Halaman 7

c) *Depth-Limited Search*

Depth-Limited Search merupakan algoritma pencarian yang menggunakan *Depth-First Search* dengan kedalaman yang dibatasi. Hal ini dilakukan untuk menghindari terjadinya *infinite search* yang dilakukan oleh algoritma *Depth-First Search*. Algoritma ini akan memperlakukan suatu simpul menjadi daun pada kedalaman tertentu.

d) *Iterative-Deepening Search*

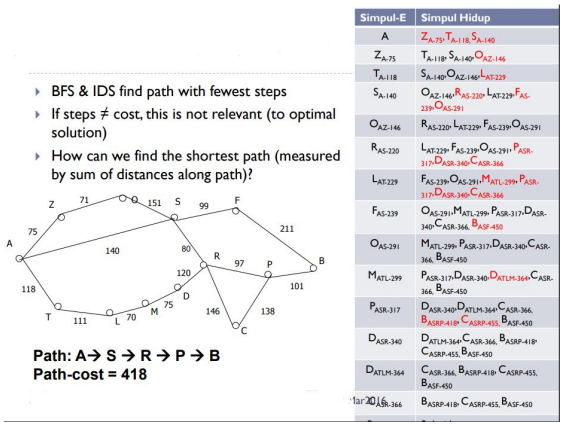
Iterative-Deepening Search merupakan algoritma pencarian yang menggunakan *Depth-Limited Search* yang dilakukan secara berkala, dimulai dari kedalaman 0, 1, 2 dan seterusnya hingga mencapai solusi yang diinginkan



Gambar 5.*Iterative-Deepening Search*
Sumber: *Slide* Kuliah Problem Solving and Search, Rinaldi Munir
Halaman 8

e) *Uniform-Cost Search*

Uniform-Cost Search merupakan algoritma pencarian yang menggunakan *Breadth-First Search*. Akan tetapi, algoritma ini akan memeriksa simpul yang memiliki *cost* terkecil yang dibutuhkan, seperti jarak antar kota, waktu yang diperlukan dari kota A ke kota B, dan lainnya tanpa memperhatikan jarak ke solusi yang diinginkan. Hal ini dilakukan dengan mengimplementasikan *priority queue* pada algoritma *Breadth-First Search*.



Gambar 6. *Uniform-Cost Search*
Sumber: *Slide* Kuliah Problem Solving and Search, Rinaldi Munir
Halaman 9

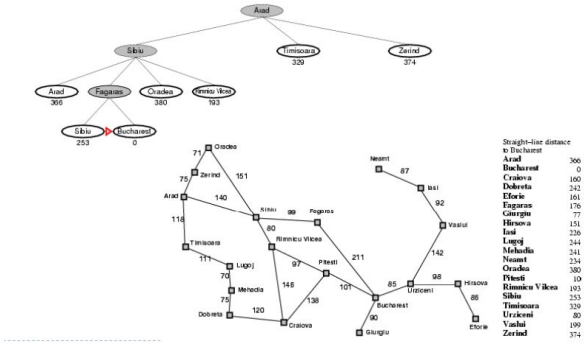
2) *Informed Search Algorithm*

Informed Search merupakan salah satu tipe algoritma pencarian yang biasa disebut *Heuristic Search*. Hal ini disebabkan oleh adanya informasi tambahan tentang kondisi eksternal yang disediakan oleh masalah dalam teknik ini. Algoritma ini dapat menemukan solusi yang lebih efisien dibandingkan dengan *Uninformed Search Algorithm*. Singkatnya, *Informed Search* adalah algoritma yang memberikan informasi spesifik tentang permasalahan yang ada disamping dari definisi dari algoritma tersebut.

Informed Search Algorithm sendiri memiliki banyak jenis, seperti:

a) *Greedy Best-First Search*

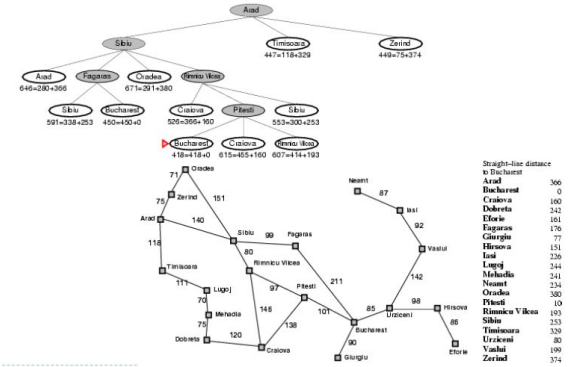
Greedy Best-First Search merupakan algoritma pencarian yang menggunakan *Breadth-First Search*. Akan tetapi, algoritma ini akan memeriksa simpul yang memiliki *cost* terkecil yang dibutuhkan ke solusi yang diinginkan. Namun, pembangkitan simpul yang dilakukan hanya melihat kepada fungsi heuristiknya saja. Hal ini dilakukan dengan mengimplementasikan *priority queue* pada algoritma *Breadth-First Search*. *Greedy Best-First Search* memiliki kelemahan pada kelengkapan dan optimalitas.



Gambar 7. *Greedy Best-First Search*
Sumber: *Slide* Kuliah Problem Solving and Search, Rinaldi Munir
Halaman 15

b) *A* Search*

A Search* merupakan algoritma gabungan dari *Uniform-Cost Search* dengan *Greedy Best-First Search*. Algoritma ini juga menggunakan *Breadth-First Search*. *Uniform-Cost Search* memeriksa simpul dengan *cost* terkecil ke simpul yang setelahnya dan *Greedy Best-First Search* memeriksa simpul dengan *cost* terkecil ke solusi, sementara *A* Search* memeriksa simpul dengan jumlah *cost* ke simpul selanjutnya dan *cost* terkecil ke solusi yang terkecil. Hal ini dilakukan dengan mengimplementasikan *priority queue* pada algoritma *Breadth-First Search*.



Gambar 8. *A* Search*
Sumber: *Slide* Kuliah Problem Solving and Search, Rinaldi Munir
Halaman 24

B. *Branch and Bound*

Branch and Bound merupakan hasil optimisasi dari algoritma *Breadth-First Search*. Algoritma ini meminimalkan atau memaksimalkan suatu fungsi objektif tanpa melanggar batasan persoalan. Algoritma ini merupakan salah satu algoritma yang sudah bisa dibilang optimal dalam mencari solusi yang diinginkan. Hal ini dikarenakan waktu yang diperlukan mendekati waktu yang diperlukan oleh *Breadth-First Search* dengan simpul yang dibangkitkan tidak sebanyak *Breadth-First Search*.

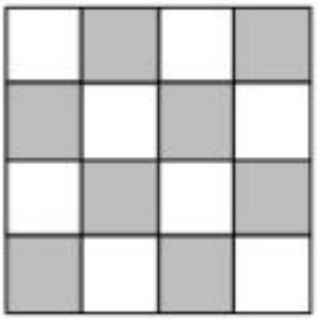
C. *Teori 15 Puzzle*

1) *Reachable Goal*

Dari semua kemungkinan *puzzle* yang terbuat, tentunya tidak semua 15 *Puzzle* bisa diselesaikan. 15 *Puzzle* bisa diselesaikan atau tidak dapat dicari dengan menggunakan rumus KURANG(i). KURANG(i) didefinisikan sebagai banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i). POSISI(i) sendiri adalah posisi ubin bernomor i pada susunan yang diperiksa. Rumus yang digunakan yaitu:

$$\sum_{i=1}^{16} Kurang(i) + X$$

dimana X merupakan letak ubin kosong berada pada *puzzle* (bernilai 1 jika berada di warna hitam dan 0 jika berada di warna putih)



Gambar 9. *Letak Ubin Kosong*
Sumber: *Slide* Kuliah Branch & Bound, Rinaldi Munir
Halaman 17

Apabila rumus menghasilkan angka yang positif, maka 15 *Puzzle* tersebut merupakan *puzzle* yang bisa diselesaikan.

2) Fungsi Heuristik

Fungsi Heuristik merupakan salah satu cara untuk membantu pencarian dengan memberitahu arah ke solusi dengan menentukan simpul mana yang lebih mendekati ke solusi yang diinginkan. Fungsi heuristik yang digunakan untuk persoalan 15 *Puzzle* adalah *Manhattan Distance*.

a) Manhattan Distance

Manhattan Distance dalam permasalahan 15 *Puzzle* merupakan fungsi heuristik yang menghitung jumlah langkah yang harus dilakukan oleh suatu ubin untuk mencapai letak yang sesuai di keadaan solusi. Cara menghitungnya adalah dengan menghitung perbedaan letak baris dan kolom antara keadaan solusi dengan keadaan *puzzle* kecuali ubin kosong.

III. Implementasi Algoritma A*

Program yang digunakan untuk mencari solusi dari 15 *Puzzle* dengan algoritma *A** merupakan program yang telah dibuat oleh akun *taueres* pada GitHub [6]. Program ini menggunakan bahasa pemrograman *Python* dan terdiri atas 6 *file*, yaitu:

1) Node.py

File Node.py digunakan untuk merepresentasikan susunan *puzzle* dari 15 *Puzzle* yang akan dicari solusinya. Nilai dari fungsi heuristik juga disimpan pada *file* ini. *Handler* untuk *puzzle* juga diatur dalam *file* ini.

2) NodeBuilder.py

File NodeBuilder.py digunakan untuk mencari simpul-simpul yang akan dibangkitkan dengan memeriksa kemungkinan-kemungkinan gerakan apa saja yang dapat dilakukan.

3) NodePool.py

File NodePool.py digunakan untuk menyimpan semua simpul yang digunakan oleh algoritma *A**. *File* ini mengimplementasikan *priority queue* untuk mengurutkan semua simpul yang dibangkitkan berdasarkan *cost* terkecil hingga terbesar.

4) AStar.py

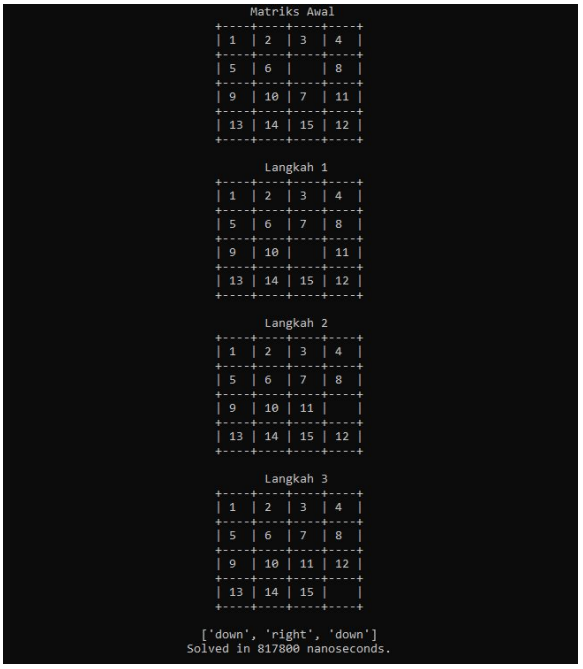
File Astar.py digunakan untuk mengimplementasikan algoritma *A** dengan mencari simpul yang akan diperiksa dengan total *cost* terkecil yang ada.

5) ManhattanDistance.py

File ManhattanDistance.py digunakan untuk menghitung semua langkah yang diperlukan oleh tiap ubin untuk menuju ke lokasi yang sesuai dengan keadaan solusi. Di *file* ini juga menentukan bagaimana keadaan solusi yang diinginkan.

6) Main.py

File Main.py digunakan sebagai *driver* dari kelima *file* yang lain. Disini juga terdapat masukkan keadaan 15 *Puzzle* yang akan dicari solusinya dan juga menampilkan langkah-langkah serta waktu yang dibutuhkan untuk mencari tiap gerakan yang dilakukan hingga solusi apabila ada atau menampilan pesan tidak ada solusi yang ada.



Gambar 10. Hasil *Output* Program *A**
Sumber: Dokumen Pribadi

IV. Implementasi Algoritma Branch and Bound

Untuk mengetahui apakah mencari solusi 15 *Puzzle* menggunakan algoritma *A** merupakan hasil yang paling optimal atau tidak, dibutuhkan suatu perbandingan dengan algoritma yang berbeda. Algoritma tersebut haruslah algoritma yang sudah bisa dibilang optimal. Dari algoritma yang ada, dibuatlah 15 *Puzzle Solver* yang menggunakan algoritma *Branch and Bound*. Program ini menggunakan bahasa pemrograman *Python* bernama *PuzzleSolver.py*. Program ini terdiri atas 3 kelas, yaitu:

1) Puzzle

Kelas *Puzzle* digunakan untuk merepresentasikan susunan *puzzle* dari 15 *Puzzle* yang akan dicari solusinya. *Handler*

`

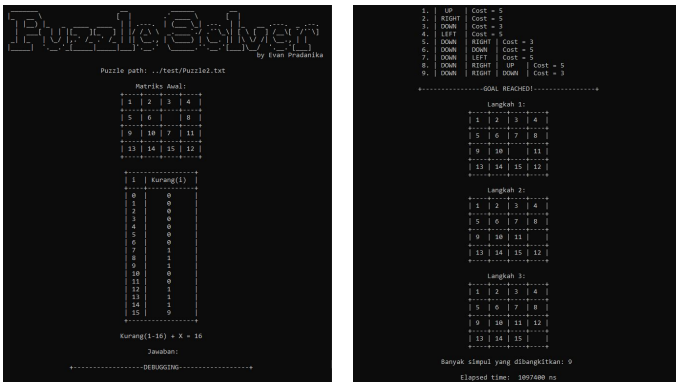
untuk *puzzle*, perhitungan rumus KURANG(i), dan perhitungan banyak perbedaan letak ubin juga diatur dalam kelas ini.

2) Solver

Kelas *Solver* digunakan untuk merepresentasikan algoritma *Branch and Bound* untuk mencari solusi yang diinginkan. Waktu yang dibutuhkan untuk mencari tiap langkah yang dibutuhkan untuk mencari solusi dan susunan setiap langkah juga diatur dalam kelas ini.

3) MainProgram

Kelas *MainProgram* digunakan sebagai *driver* untuk menjalankan program ini. Kelas ini akan memanggil fungsi-fungsi dari 2 kelas diatas untuk menampilkan judul, susunan *puzzle*, hasil rumus KURANG(i), waktu yang dibutuhkan, dan tiap susunan *puzzle* dari langkah pertama hingga solusi.



Gambar 11. Hasil *Output* Program *Branch and Bound*
Sumber: Dokumen Pribadi

V. Uji Coba

Pengujian dilakukan menggunakan 5 *test case* yang telah dibuat oleh penulis. Apabila waktu yang dibutuhkan melebihi 3 menit, akan diasumsikan bahwa program tersebut tidak bisa menemukan solusi yang diinginkan.

Kelima *test case* tersebut berupa seperti berikut:

1) Puzzle1.txt

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

2) Puzzle2.txt

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

3) Puzzle3.txt

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

4) Puzzle4.txt

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

5) Puzzle5.txt

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

Tabel 1. Algoritma *A* vs Branch and Bound*

Nama Puzzle	**Banyak Langkah**	**Total Waktu (ns)**	
		A*	***Branch and Bound***
Puzzle1.txt	3	817800	1097400
Puzzle2.txt	9	1699300	2736000
Puzzle3.txt	12	2556600	4492400
Puzzle4.txt	16	43396100	-
Puzzle5.txt	34	6063708300	-

`

Dapat dilihat dari hasil tabel diatas bahwa algoritma *A** dapat menemukan solusi dengan waktu yang lebih singkat dibandingkan dengan algoritma *Branch and Bound*. Kedua algoritma menghasilkan solusi yang sama. Untuk *Puzzle4.txt* dan *Puzzle5.txt*, algoritma *Branch and Bound* membutuhkan waktu yang sangat lama, melebihi batas 3 menit sehingga diasumsikan bahwa 15 *Puzzle Solver* dengan algoritma *Branch and Bound* tidak dapat menemukan solusi dari permasalahan *Puzzle4.txt* dan *Puzzle5.txt*.

VI. Simpulan

Algoritma *A** merupakan algoritma pencarian yang menggunakan fungsi heuristik untuk menambah informasi terhadap permasalahan yang ada. Algoritma *A** juga bisa digunakan untuk mencari solusi dari permasalahan 15 *Puzzle*. Untuk optimalisasinya bisa dilihat dari perbandingan kecepatan antara penggunaan algoritma *A** dengan algoritma *Branch and Bound* yang dibutuhkan untuk menemukan langkah-langkah dari susunan *puzzle* hingga mencapai solusi.

Oleh karena itu, algoritma *A** adalah algoritma pencarian yang paling optimal dibandingkan dengan algoritma-algoritma pencarian lainnya yang sudah penulis jelaskan dikarenakan algoritma ini dapat menemukan solusi dari permasalahan yang ada beserta langkah-langkah susunannya dengan waktu yang singkat sehingga algoritma *A** sangat bisa dan dianjurkan untuk diimplementasikan kepada permasalahan 15 *Puzzle*.

VII. Video Link Youtube

https://www.youtube.com/watch?v=FOIX1UR5JxA - Algoritma A* dengan Manhattan Distance

VIII. Ucapan Terima Kasih

Puji syukur penulis panjatkan pada hadirat Allah SWT. atas rahmat dan kemudahan yang diberikan kepada penulis, selama penulis mengerjakan makalah yang berjudul "Pemanfaatan Algoritma A* dalam Penyelesaian 15 Puzzle Game". Banyak pihak yang telah membantu penulis dalam penyusunan makalah ini. Oleh karena itu, penulis sangat bersyukur dan ingin menyampaikan rasa terima kasih kepada Dr. Ir. Rinaldi Munir, MT., selaku Dosen Mata Kuliah IF-2211 Strategi Algoritma dan teman-teman seperjuangan yang telah memberikan banyak masukan, referensi, dan dukungan.

IX. Referensi

[1] R. Munir, *Slide Problem Solving and Search (2020)*, Bandung.

[2] R. Munir, *Slide Branch & Bound (2018)*, Bandung.

[3] https://www.sciencedirect.com/topics/engineering/path-planning. Diakses pada tanggal 3 Mei 2020 pukul 13.30 WIB.

[4] https://socs.binus.ac.id/2013/04/23/uninformed-search-dan-informed-search/. Diakses pada tanggal 3 Mei 2020 pukul 14.00 WIB..

[5] https://xlinux.nist.gov/dads/HTML/manhattanDistance.html.Diakses pada tanggal 3 Mei 2020 pukul 15.00 WIB.

[6] https://github.com/taueres/a-star-15-puzzle-solver. Diakses pada tanggal 3 Mei 2020 pukul 16.30 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, 3 Mei 2020

Evan Pradanika - 13518126

`