TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA

PENYELESAIAN PERSOALAN 15-PUZZLE DENGAN ALGORITMA BRANCH AND BOUND

Disusun oleh :

Christine Hutabarat (13520005)

TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG 2022

DAFTAR ISI

I. ALGORITMA BRANCH AND BOUND

Algoritma *branch and bound* adalah suatu algoritma yang digunakan untuk menyelesaikan persoalan optimasi, yaitu suatu algoritma yang mengutamakan nilai minimal/maksimal dari fungsi objektif persoalan terkait. Algoritma ini memiliki karakter yang merupakan gabungan antara karakteristik algoritma *breadth-first search* dan algoritma *least cost search*. Algoritma *branch and bound* bergantung pada taksiran 'harga' solusi relatif terhadap suatu simpul yang umum disebut sebagai '*cost*'. Untuk menentukan urutan simpul yang akan dikunjungi pada langkah berikutnya, digunakan struktur data *priority queue*, di mana simpul dengan *cost* terendah akan mendapatkan prioritas yang lebih tinggi dan diletakkan di bagian awal antrian pada kasus minimasi, sementara simpul dengan *cost* tertinggi akan mendapatkan prioritas yang lebih tinggi pada kasus maksimasi. Selain dengan karakteristiknya yang mempertimbangkan *cost* dari suatu simpul, algoritma *branch and bound* juga memangkas atau membunuh simpul yang tidak mengarah ke solusi. Hal ini mirip seperti penghapusan simpul pada algoritma *backtracking*, namun pada algoritma *backtracking*, nilai *cost* tidak ada, sehingga tidak ada batasan dari ekspansi simpul. Sementara itu, pada algoritma *branch and bound*, nilai *cost* setiap simpul dipertimbangkan untuk kemudian salah satunya dipilih sebagai yang terbaik dan dilakukan ekspansi dan jalur yang tidak mengarah pada solusi akan dipangkas. Secara garis besar, algoritma *branch and bound* diawali dengan memasukkan simpul akar ke dalam antrian, untuk kemudian diperiksa dan dihapus dari antrian. Jika simpul akar adalah simpul solusi, maka pencarian dihentikan. Jika bukan, maka simpul akar akan diperluas, dan anak-anaknya dimasukkan ke dalam antrian. Iterasi akan dihentikan jika antrian kosong. Jika antrian tidak kosong, simpul anak dengan *cost* terkecil/terbesar akan diperiksa dan dihapus dari antrian. Iterasi berhenti jika simpul yang sedang diperiksa adalah simpul solusi, dan dilanjutkan dengan memasukkan anak-anak dari simpul yang sedang diperiksa ke dalam antrian. Langkah-langkah ini diulangi terus menerus hingga simpul solusi ditemukan atau antrian kosong.

Persoalan 15-Puzzle adalah suatu persoalan di mana terdapat 15 buah ubin dan satu ubin kosong yang tersusun secara simetris, dan masing-masing ubin yang tidak kosong memiliki nilai unik 1-15. Tujuan dari permainan adalah menggeser ubin-ubin dengan memanfaatkan ruang yang diberikan ubin kosong, hingga terbentuk suatu konfigurasi dari ubin yang diharapkan. Ubin hanya dapat bergerak dengan cara 'digeser', yang berarti bahwa ubin hanya dapat bergerak ke kanan, kiri, atas, dan bawah. Pergeseran ubin akan menghasilkan percabangan untuk kandidat jalur menuju solusi. Jika didefinisikan suatu fungsi Kurang(i) yaitu banyaknya ubin bernomor j sedemikian sehingga j < i namun posisi(i) < posisi(j), dan variabel X yang bernilai 1 jika ubin kosong berada pada baris ganjil dan kolom genap, bernilai 0 jika sebaliknya, maka suatu teorema mengenai persoalan ini menyatakan bahwa status tujuan dari ubin-ubin akan didapatkan hanya jika untuk *i* bernilai 1-15, jumlah keseluruhan Kurang(i) ditambah X bernilai genap. Jika persoalan 15-Puzzle diselesaikan dengan algoritma *branch and bound*, maka fungsi objektif dari persoalan adalah mencapai status tujuan dengan jumlah pergeseran seminimal mungkin. Karena dari status awal letak simpul solusi umumnya tidak diketahui, maka digunakan taksiran terhadap nilai *cost*, yang didefinisikan dengan persamaan berikut.

$$\hat{c}(i) = \hat{f}(i) + \hat{g}(i) \tag{1}$$

$\hat{c}(i)$ = ongkos untuk simpul *i*
$\hat{f}(i)$ = jumlah langkah dari simpul akar ke simpul *i*
$\hat{g}(i)$ = taksiran harga untuk mencapai simpul tujuan dari simpul *i*

Nilai *cost* merupakan nilai pembatas dalam algoritma *branch and bound* yang digunakan, di mana nilai dari $\hat{c}(i)$ terkecil akan menjadi prioritas ekspansi simpul berikutnya. Dengan menggunakan struktur *priority queue* untuk menyimpan taksiran nilai simpul, maka secara alami akan diproses

terlebih dahulu simpul dengan taksiran nilai *cost* terkecil, sementara simpul dengan taksiran nilai *cost* yang lebih besar akan berada di bagian belakang antrian.

II. IMPLEMENTASI

Program 15-Puzzle Solver ditulis dengan menggunakan bahasa pemrograman python dan sebagiannya diimplementasikan dengan pendekatan pemrograman berorientasi objek. Keseluruhan kode program beserta contoh kasus yang dapat digunakan untuk melakukan percobaan terdapat dalam repository GitHub dengan alamat :

https://github.com/chryes220/Tucil3-Stima.git

Kode program berada pada folder *src* dan program tersusun dari tiga buah file python, yaitu puzzle.py, solver.py, dan main.py. File puzzle.py berisi kelas Puzzle, yang merepresentasikan suatu status dari persoalan 15-Puzzle. File solver.py berisi kelas PriorityQueue, yaitu kelas antrian dengan prioritas yang digunakan untuk menentukan status mana yang akan diperiksa berikutnya. Selain itu, terdapat juga kelas Solver yang berisi struktur-struktur data yang diperlukan selama penyelesaian, serta fungsi-fungsi dan prosedur yang juga diperlukan untuk menyelesaikan persoalan. Kode program utama terdapat pada file main.py, yang berisi langkah pembacaan nama file masukan, pengubahan isi file ke dalam bentuk puzzle yang dikenali yaitu sebagai suatu objek dari kelas Puzzle, hingga pemanggilan fungsi penyelesaian puzzle.

Untuk menyelesaikan puzzle, harus dibuat terlebih dahulu objek dari kelas Solver, dengan menggunakan parameter pada konstruktornya yaitu puzzle yang akan diselesaikan. Setelah objek berhasil dibuat, metode *solve()* milik kelas Solver akan dipanggil untuk mengisi atribut-atribut yang terkait dengan solusi dari persoalan, dan menampilkan langkah serta hasil algoritma penyelesaian. Sebelum penyelesaian dimulai oleh metode *solve()*, akan diperiksa terlebih dahulu apakah puzzle dapat diselesaikan atau tidak. Jika puzzle dapat diselesaikan, maka algoritma *branch and bound* akan dimulai sementara jika tidak, program akan menampilkan pesan bahwa puzzle tidak dapat diselesaikan pada layar, dan program dihentikan.

Program menerima masukan berupa file teks yang terdiri dari empat baris, dengan masing-masing baris berisi empat angka unik bernilai di antara 1 hingga 16, dipisahkan dengan sebuah spasi. Berikut ini adalah contoh isi dari file teks yang diterima program.

1 2 3 4

5 6 11 7

14 16 10 8

9 13 15 12

Angka 1 hingga 16 menandakan posisi di mana ubin seharusnya berada pada status tujuan, sehingga ubin dengan angka 16 merepresentasikan ubin kosong, yang nilai angkanya tidak akan ditampilkan pada penampilan status puzzle.

Adapun algoritma *branch and bound* dimulai oleh metode *solve()* dengan langkah pertama yaitu memasukkan simpul akar pada antrian, menandainya sebagai status yang pernah dialami, kemudian menghapusnya dari antrian serta memeriksanya. Setelah pemeriksaan simpul, iterasi akan dilanjutkan dengan menghapus dan memproses simpul pada antrian berikutnya. Iterasi dihentikan jika nilai dari atribut *fin* telah bernilai *True* yang menandakan bahwa simpul tujuan telah dicapai. Pemrosesan simpul dilakukan oleh metode *branchBound()* pada file solver.py. Metode ini menerima parameter berupa puzzle yang akan diproses, serta sebuah bilangan bulat positif yang menandakan jumlah langkah menuju simpul puzzle yang sedang diperiksa dari simpul akar. Dalam algoritma metode ini, hal pertama yang dilakukan adalah memeriksa apakah status puzzle merupakan status tujuan. Jika status puzzle adalah status tujuan, maka nilai atribut *fin* akan diubah dari *False* menjadi *True*, dan urutan langkah-

langkah yang diambil untuk mencapai status tersebut akan disimpan dalam atribut *sequence*. Namun jika ternyata status puzzle bukanlah status tujuan, akan dibangkitkan simpul-simpul anak dengan masing-masing anak adalah hasil pergeseran ubin kosong ke empat arah yang berbeda, yaitu atas, kanan, bawah, dan kiri. Urutan tersebut juga merupakan urutan arah dari pembangkitan simpul anak yang baru. Jika konfigurasi ubin dari simpul anak sudah pernah dilalui sebelumnya, maka akan dibandingkan nilai *cost* dari simpul baru dengan simpul lama. Jika nilai *cost* dari simpul baru lebih besar daripada *cost* dari simpul lama, maka akan dikembalikan nilai *True*. Sementara itu, jika nilai *cost* dari simpul baru lebih kecil daripada *cost* simpul lama, maka akan dicatat nilai *cost* terkecil untuk konfigurasi ubin terkait. Setelah simpul anak dibuat dan dibangkitkan, nilai *cost* dari simpul anak dihitung dengan memanfaatkan metode *countCost()* milik kelas Puzzle. Setelah itu, status dari simpul anak akan ditandai sebagai status yang pernah dialami dan simpul anak dimasukkan ke dalam antrian. Algoritma dari metode *branchBound()* dihentikan dan jalannya program dikembalikan ke metode *solve()*. Yang berikutnya akan dilakukan program hanyalah menghitung waktu berjalannya program, menampilkan status awal puzzle, menampilkan nilai Kurang(i) untuk setiap ubin pada status awal, kemudian menampilkan waktu berjalannya algoritma *branch and bound* dan juga status dari puzzle untuk setiap langkah menuju simpul tujuan.

III. HASIL PERCOBAAN

Dalam folder test pada repository, telah terdapat lima contoh file masukan yang diterima oleh program. Tiga diantaranya, yaitu tc_solvable_1.txt, tc_solvable_2.txt, dan tc_solvable_3.txt merupakan contoh status awal puzzle yang dapat diselesaikan. Sementara itu, dua file sisanya merupakan contoh status awal puzzle yang tidak dapat diselesaikan. Tabel 3.1 menampilkan isi dari masing-masing file percobaan serta hasil penyelesaiannya menggunakan program yang telah dibuat.

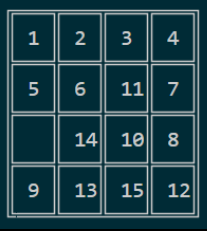
Tabel 3.1. Hasil Percobaan Setiap File *Test Case*

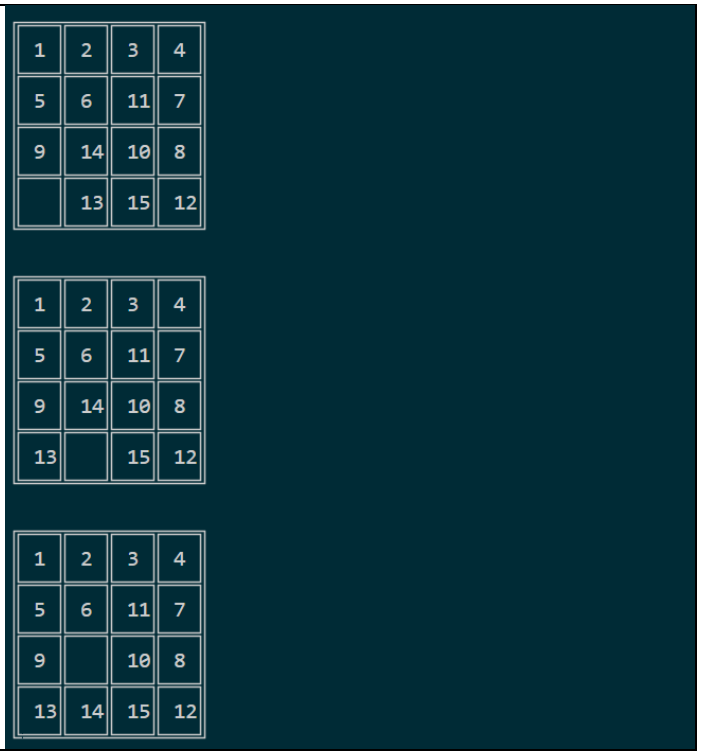
Nama File	Isi File	Hasil Penyelesaian
tc_solvable_1.txt	1 2 3 4 5 6 16 8 9 10 7 11 13 14 15 12	Use randomized puzzle? [y/n] n Input file name : tc_solvable_1.txt Initial state | 1 | 2 | 3 | 4 | | 5 | 6 | | 8 | | 9 | 10 | 7 | 11 | | 13 | 14 | 15 | 12 | Kurang(1) = 0 Kurang(2) = 0 Kurang(3) = 0 Kurang(4) = 0 Kurang(5) = 0 Kurang(6) = 0 Kurang(7) = 0 Kurang(8) = 1 Kurang(9) = 1 Kurang(10) = 1 Kurang(11) = 0 Kurang(12) = 0 Kurang(13) = 1 Kurang(14) = 1 Kurang(15) = 1 Kurang(16) = 9

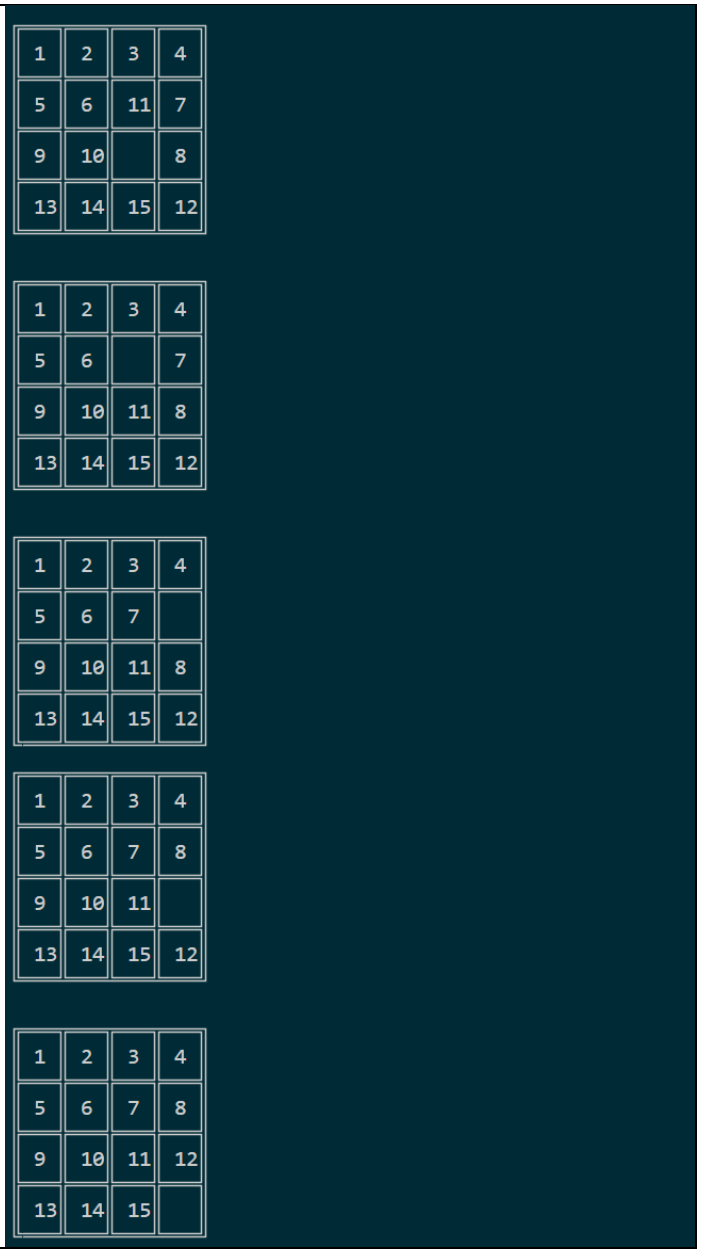
<table>
<tr>
<td></td>
<td></td>
<td>
Puzzle is solvable with value of Sum(Kurang(i)) + X = 16

Solution found in 0.003917694091796875 s
Total node : 10

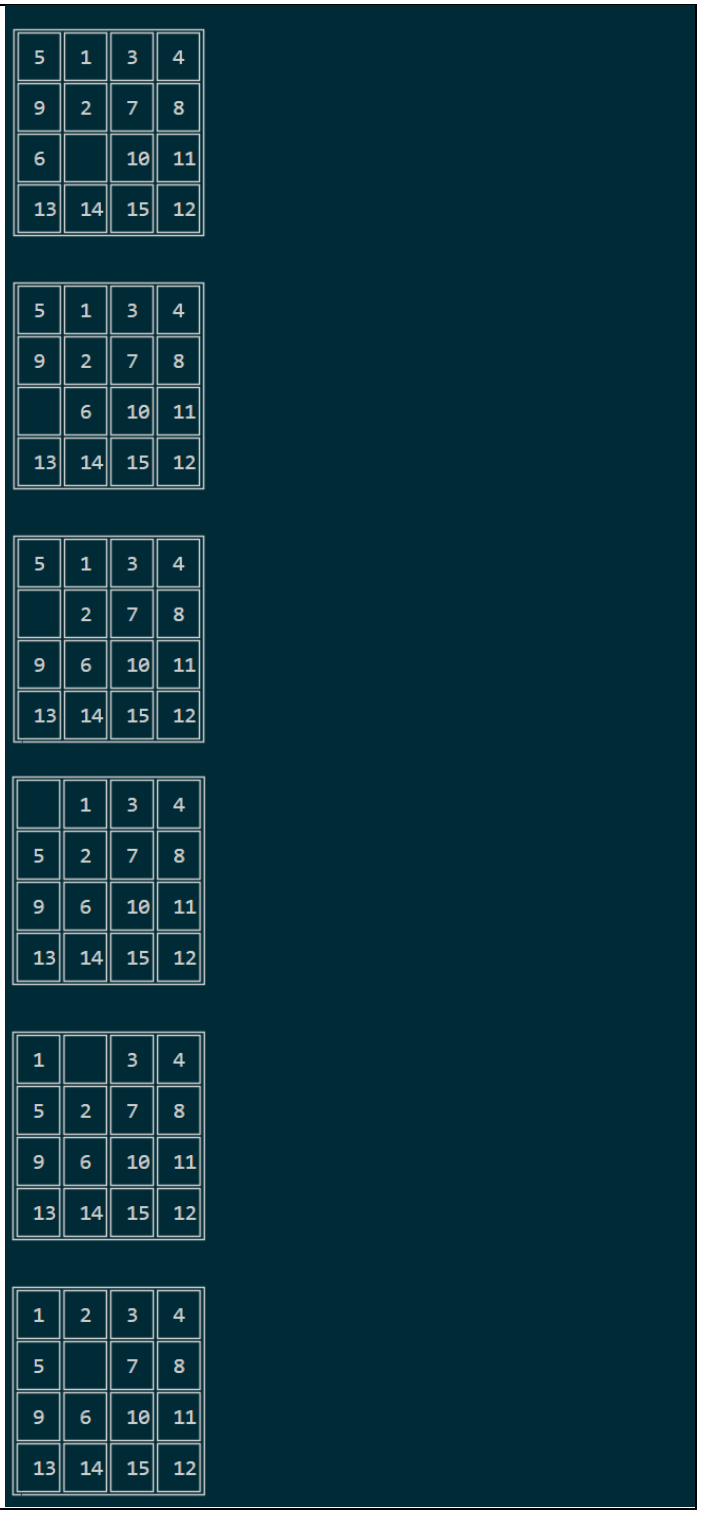
Move sequence :
<table>
<tr><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td></td><td>8</td></tr>
<tr><td>9</td><td>10</td><td>7</td><td>11</td></tr>
<tr><td>13</td><td>14</td><td>15</td><td>12</td></tr>
</table>
<table>
<tr><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr><td>9</td><td>10</td><td></td><td>11</td></tr>
<tr><td>13</td><td>14</td><td>15</td><td>12</td></tr>
</table>
<table>
<tr><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr><td>9</td><td>10</td><td>11</td><td></td></tr>
<tr><td>13</td><td>14</td><td>15</td><td>12</td></tr>
</table>
<table>
<tr><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr><td>9</td><td>10</td><td>11</td><td>12</td></tr>
<tr><td>13</td><td>14</td><td>15</td><td></td></tr>
</table>
</td>
</tr>
<tr>
<td>tc_solvable_2.txt</td>
<td>1 2 3 4
5 6 11 7
14 16 10 8
9 13 15 12</td>
<td></td>
</tr>
</table>

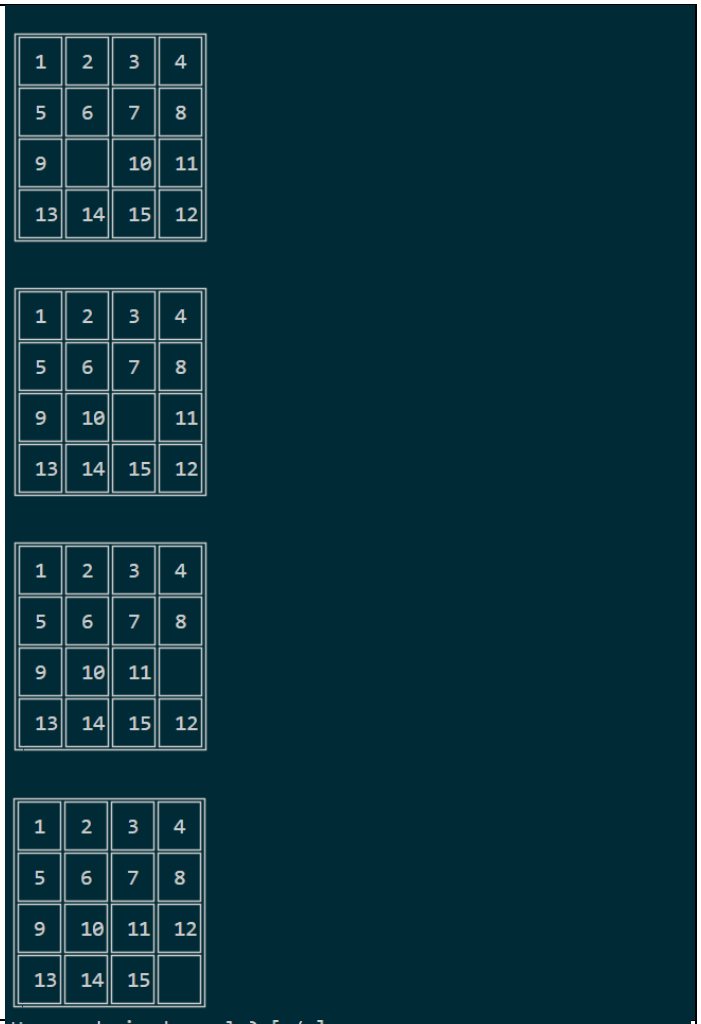
		Use randomized puzzle? [y/n] n Input file name : tc_solvable_2.txt Initial state  Kurang(1) = 0 Kurang(2) = 0 Kurang(3) = 0 Kurang(4) = 0 Kurang(5) = 0 Kurang(6) = 0 Kurang(7) = 0 Kurang(8) = 0 Kurang(9) = 0 Kurang(10) = 2 Kurang(11) = 4 Kurang(12) = 0 Kurang(13) = 1 Kurang(14) = 5 Kurang(15) = 1 Kurang(16) = 6 Puzzle is solvable with value of Sum(Kurang(i)) + X = 20 Solution found in 0.024113178253173828 s Total node : 39 Move sequence :  

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

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

tc_solvable_3.txt	5 1 3 4 9 2 16 8 6 10 7 11 13 14 15 12	Use randomized puzzle? [y/n] n Input file name : tc_solvable_3.txt Initial state | 5 | 1 | 3 | 4 | | 9 | 2 | | 8 | | 6 | 10 | 7 | 11 | | 13 | 14 | 15 | 12 | Kurang(1) = 0 Kurang(2) = 0 Kurang(3) = 1 Kurang(4) = 1 Kurang(5) = 4 Kurang(6) = 0 Kurang(7) = 0 Kurang(8) = 2 Kurang(9) = 4 Kurang(10) = 1 Kurang(11) = 0 Kurang(12) = 0 Kurang(13) = 1 Kurang(14) = 1 Kurang(15) = 1 Kurang(16) = 9 Puzzle is solvable with value of Sum(Kurang(i)) + X = 26 Solution found in 0.07813096046447754 s Total node : 73 Move sequence : | 5 | 1 | 3 | 4 | | 9 | 2 | | 8 | | 6 | 10 | 7 | 11 | | 13 | 14 | 15 | 12 | | 5 | 1 | 3 | 4 | | 9 | 2 | 7 | 8 | | 6 | 10 | | 11 | | 13 | 14 | 15 | 12 |

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

		
tc_not_solvable_1.txt	6 4 15 13 5 9 14 12 8 7 11 1 10 16 3 2	Use randomized puzzle? [y/n] n Input file name : tc_not_solvable_1.txt Initial state 6 4 15 13 5 9 14 12 8 7 11 1 10 3 2 Kurang(1) = 0 Kurang(2) = 0 Kurang(3) = 1 Kurang(4) = 3 Kurang(5) = 3 Kurang(6) = 5 Kurang(7) = 3 Kurang(8) = 4 Kurang(9) = 5 Kurang(10) = 2 Kurang(11) = 4 Kurang(12) = 7 Kurang(13) = 10 Kurang(14) = 8 Kurang(15) = 12 Kurang(16) = 2 Puzzle is not solvable with value of Sum(Kurang(i)) + X = 69

tc_not_solvable_2.txt	8 6 16 5 10 11 4 7 3 14 15 9 12 1 2 13	Use randomized puzzle? [y/n] n Input file name : tc_not_solvable_2.txt Initial state | 8 | 6 | | 5 | | 10 | 11 | 4 | 7 | | 3 | 14 | 15 | 9 | | 12 | 1 | 2 | 13 | Kurang(1) = 0 Kurang(2) = 0 Kurang(3) = 2 Kurang(4) = 3 Kurang(5) = 4 Kurang(6) = 5 Kurang(7) = 3 Kurang(8) = 7 Kurang(9) = 2 Kurang(10) = 6 Kurang(11) = 6 Kurang(12) = 2 Kurang(13) = 0 Kurang(14) = 5 Kurang(15) = 5 Kurang(16) = 13 Puzzle is not solvable with value of Sum(Kurang(i)) + X = 63

IV. EVALUASI

Untuk kasus-kasus sederhana yang tidak memerlukan banyak langkah penyelesaian, program berhasil menyelesaikan persoalan 15-Puzzle dengan baik. Meskipun begitu, dengan menggunakan algoritma *branch and bound*, jumlah simpul yang dibangkitkan cukup banyak sehingga untuk kasus 15-Puzzle yang lebih rumit, program akan menyelesaikan persoalan dalam waktu yang sangat lama. Hal ini dapat dibuktikan dengan menjalankan percobaan pada puzzle yang telah di-*random.* Keseluruhan hasil percobaan dirangkum dalam tabel 4.1.

Tabel 4.1. Evaluasi Program

Poin	Ya	Tidak
1. Program berhasil dikompilasi	✓	
2. Program berhasil *running*	✓	
3. Program dapat menerima input dan menuliskan output	✓	
4. Luaran sudah benar untuk semua data uji	✓	
5. Bonus dibuat		✓