

# **IF2211 Strategi Algoritma**

Algoritma Penyelesaian Problem *15 Puzzle* dengan Strategi *Branch & Bound*

Tugas Kecil III



Oleh:

**Matthew Kevin Amadeus 13518035**

**PROGRAM STUDI SARJANA INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2020**

## **DAFTAR ISI**

DAFTAR ISI	2
BAB I	
ALGORITMA DAN KOMPLEKSITAS	3
BAB II	
IMPLEMENTASI PROGRAM	6
BAB III	
SCREENSHOT PROGRAM	17
LAMPIRAN	19

## BAB I PENJELASAN ALGORITMA

### 1. Definisi 15 Puzzle

15 Puzzle adalah suatu jenis teka-teki yang sering ditemui. Teka-teki ini sederhana, yaitu teka-teki yang disusun dalam *grid* berukuran 4x4 yang masing-masing selnya terisi angka dari 1-15. Teka-teki tersebut dikatakan terpecahkan(*solved*) apabila berada dalam posisi yang terurut dari 1-15 dengan bagian yang kosong di ujung bawah kanan. Teka-teki ini hanya bisa digeser tiap selnya ke bagian yang kosong.



Gambar 1.1 : Contoh 15 Puzzle  
(Sumber: Wikipedia)

### 2. Algoritma *Branch and Bound* untuk Menyelesaikan 15 Puzzle

*Branch and Bound* adalah salah satu jenis strategi algoritma yang digunakan untuk menjabarkan *state* dari suatu persoalan bila diketahui *state* tujuan dari persoalan tersebut. Tergantung dari fungsi heuristiknya, seberapa efisien algoritma ini bekerja bisa berbeda. Semakin sedikit *state* yang diibangkitkan, semakin efisien algoritma ini bekerja. *Branch and Bound* ini bisa menghemat dalam membangkitkan *state*-nya dengan cara memilih *state* saat ini yang memiliki ongkos paling kecil, dengan harapan akan mencapai *state* tujuan dengan cepat.

Untuk menyelesaikan 15 Puzzle, tepat digunakan algoritma *Branch and Bound*, karena dalam 15 Puzzle sudah jelas apa yang menjadi *state* tujuan, yaitu *state* akhir di mana posisi angka-angka dalam 15 Puzzle tersebut sudah terurut. Dalam algoritma ini juga yang diperhatikan adalah kotak kosongnya sebagai penanda. Langkah-langkah penyelesaian yang perlu dilakukan adalah sebagai berikut:

1. Menentukan *state* awal dari teka-teki
2. Menentukan apakah *state* tersebut bisa diselesaikan dengan menghitung jumlah inversi yang ada serta melihat posisi kotak kosongnya.
3. Menentukan *state* yang bisa dicapai dari *state* saat ini. Jika tidak mungkin, tidak perlu membangkitkan *state*-nya. *State* yang mungkin dicapai adalah seperti:

- a. Apakah kotak kosong bisa berpindah ke atas atau tidak.
    - i. Apakah kotak kosong bisa berpindah ke kiri atau tidak.
  - b. Apakah kotak kosong bisa berpindah ke kanan atau tidak.
  - c. Apakah kotak kosong bisa berpindah ke bawah atau tidak
4. Menghitung ongkos taksiran, yang didefinisikan sebagai berikut:

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

Taksiran tersebut merupakan penjumlahan dari ongkos perpindahan antar state (yang dalam kasus ini bernilai 1 karena terdapat 1 pergeseran) dan perkiraan ongkos dari state itu ke state akhir. Dalam kasus ini, dipilih perkiraan yang bernilai sama dengan jumlah sel yang berisi kotak yang salah.

5. Ulangi dari langkah ke 3 untuk *state* yang dipilih.
6. Hentikan proses apabila sudah mencapai *state* akhir.

## BAB II

### SCREENSHOT PROGRAM

Berikut ini adalah implementasi dari program yang dibuat oleh penulis. Penulis membuat program ini dengan bahasa Python dengan prinsip OOP yang tersedia dalam Python. Selain itu, spesifikasi komputer yang digunakan oleh penulis adalah sebagai berikut:

- a. Processor : Intel(R) Core(TM) i7-8550U @ 1.80GHz
- b. Memory : 8192 MB
- c. GPU : Intel(R) UHD Graphics 620 dan NVIDIA GeForce MX150

Berikut ini adalah hasil percobaan program untuk masukan yang berbeda-beda.

1. Untuk tampilan menu help

```
D:\_University\Sem. 4\Stima\Puzzle>python main.py -h
usage: python main.py [-h] [-sh] [-md] filename

Generate solution to 15 puzzle. Can be extended to n^2 - 1 puzzles.

positional arguments:
  filename              filename of puzzle to be solved

optional arguments:
  -h, --help            show this help message and exit
  -sh, --shorthand      print out the shorthand solution only
  -md, --manhattandist  calculate solution using Manhattan distance
```

Gambar 3.1 : Tampilan menu help

2. Untuk testcase 1

```
D:\_University\Sem. 4\Stima\Puzzle>python main.py -sh solveable_01.txt
 1  2  4  7
 5  6  #  3
 9 11 12  8
13 10 14 15

Inversions: 21
Parity: 1
Total: 22 (even)
Puzzle is solveable.

Total moves: 11
R U L D R D L L D R R Solved
70 nodes generated
15.625 ms taken
```

Gambar 3.2 : Tampilan hasil testcase 1

3. Untuk testcase 2

```

D:\_University\Sem. 4\Stima\Puzzle>python main.py -sh solveable_02.txt
  1  2 12  3
  5  6  8  4
13  9 11 15
10  #  7 14

Inversions: 26
Parity: 0
Total: 26 (even)
Puzzle is solveable.

Total moves: 20
L U R R U U R D L D D R U U L D L D R R Solved
6566 nodes generated
55375.0 ms taken

```

Gambar 3.3 : Tampilan hasil testcase 2

4. Untuk testcase 3

```

D:\_University\Sem. 4\Stima\Puzzle>python main.py -sh solveable_03.txt
  1  2  3  4
  5  6  7  8
11 12 15 14
10  9 13  #

Inversions: 12
Parity: 0
Total: 12 (even)
Puzzle is solveable.

Total moves: 16
U L L L D R R R U L L L D R R R Solved
1199 nodes generated
2296.875 ms taken

```

Gambar 3.4 : Tampilan hasil testcase 3

5. Untuk testcase 4

```

D:\_University\Sem. 4\Stima\Puzzle>python main.py -sh unsolveable_01.txt
  2  1  4  7
  5  6  #  3
  9 11 12  8
13 10 14 15

Inversions: 22
Parity: 1
Total: 23 (odd)
Puzzle is unsolveable.

```

Gambar 3.5 : Tampilan hasil testcase 4

6. Untuk testcase 5

```
D:\_University\Sem. 4\Stima\Puzzle>python main.py -sh unsolveable_02.txt
  1   2  14   3
  5   6   4   7
15  10  11   #
  9  13   8  12

Inversions: 30
Parity: 1
Total: 31 (odd)
Puzzle is unsolveable.
```

Gambar 3.6 : Tampilan hasil testcase 5

### BAB III

#### KESIMPULAN

Dari hasil percobaan beberapa testcase, bisa disimpulkan semakin “jauh” atau kompleks suatu konfigurasi dari 15 Puzzle, akan lebih lambat dalam penyelesaiannya, karena jauh lebih banyak *state* yang dibangkitkan. Semakin kompleks suatu konfigurasi mengakibatkan konsumsi sumber daya yang besar, sehingga sebaiknya diperlukan metode lain untuk menyelesaikan permasalahan ini.

Salah satu cara mempercepat program ini adalah dengan mengganti fungsi heuristik atau fungsi taksiran  $\hat{g}(i)$ , karena dengan memilih fungsi yang tepat program bisa diselesaikan jauh lebih cepat. Dalam program ini, digunakan fungsi heuristik dengan menghitung jumlah sel yang terletak secara salah, atau biasa juga disebut *misplaced tiles*. Kelemahan fungsi ini adalah yakni fungsi ini tidak memperhitungkan seberapa jauh sel tersebut terletak dari sel yang seharusnya.

Sehingga, bisa digunakan fungsi heuristik lain, yaitu dengan menggunakan *Manhattan Distance*. Dengan *Manhattan Distance*, jarak pun menjadi perhitungan untuk memilih simpul mana yang akan dibandingkan, sehingga pencarian solusi menjadi lebih efisien. Dalam program ini juga diberikan penelusuran dengan *Manhattan Distance* sebagai pembanding.

```
D:\_University\Sem. 4\Stima\Puzzle>python main.py -sh -md solveable_01.txt
 1  2  4  7
 5  6  #  3
 9 11 12  8
13 10 14 15

Inversions: 21
Parity: 1
Total: 22 (even)
Puzzle is solveable.

Total moves: 11
R U L D R D L L D R R Solved
40 nodes generated
0.0 ms taken
```

Gambar 3.1 Testcase 1 dengan taksiran *Manhattan Distance*



```

D:\_University\Sem. 4\Stima\Puzzle>python main.py -sh -md solveable_02.txt
 1  2 12  3
 5  6  8  4
13  9 11 15
10  #  7 14

Inversions: 26
Parity: 0
Total: 26 (even)
Puzzle is solveable.

Total moves: 20
L U R R U U R D L D D R U U L D L D R R Solved
192 nodes generated
78.125 ms taken

```

Gambar 3.2 Testcase 2 dengan taksiran *Manhattan Distance*

```

D:\_University\Sem. 4\Stima\Puzzle>python main.py -sh -md solveable_03.txt
 1  2  3  4
 5  6  7  8
11 12 15 14
10  9 13  #

Inversions: 12
Parity: 0
Total: 12 (even)
Puzzle is solveable.

Total moves: 16
U L L L D R R R U L L L D R R R Solved
45 nodes generated
0.0 ms taken

```

Gambar 3.3 Testcase 3 dengan taksiran *Manhattan Distance*

Terdapat beberapa jenis fungsi taksiran lainnya yang lebih efisien. Jadi, dalam penyelesaian masalah ini memang terdapat banyak cara, tapi sebaiknya dipilih cara yang paling optimal dalam menyelesaikan masalah ini, contohnya dengan menggunakan algoritma A\*.

## LAMPIRAN

Poin	Ya	Tidak
Program berhasil dikompilasi	✓	
Program <i>running</i>	✓	
Program dapat menerima input dan menuliskan output	✓	
Luaran sudah benar untuk semua n	✓	

## **REFERENSI**

Kim, M. (2020). Michael Kim | Solving the 15 Puzzle. Dikunjungi 25 Maret 2020, dari <https://michael.kim/blog/puzzle>