LAPORAN TUGAS KECIL 3

IF2211 STRATEGI ALGORITMA

Disusun oleh:

Hansel Valentino Tanoto

13520046

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

A. ALGORITMA *BRANCH AND BOUND*

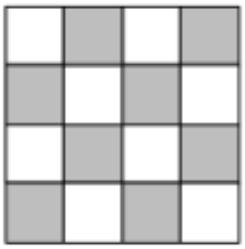
Algoritma *Branch* and *Bound* (B&B) pada umumnya digunakan pada persoalan-persoalan optimasi yaitu dengan cara meminimalkan atau memaksimalkan fungsi objektif dengan tetap tidak melanggar *constraint* tertentu. B&B dapat dikatakan merupakan penggabungan dari algoritma BFS (*Breadth First Search*) dan *Least Cost Search*, dimana setiap simpul pada pohon pencarian B&B memiliki suatu cost / biaya dan simpul ekspannya akan dipilih berdasarkan nilai *cost*-nya tersebut (maksimasi/minimasi), tidak lagi secara FIFO (*First In First Out*). Sama seperti algoritma *backtracking*, simpul yang tidak mengarah ke solusi akan "dipangkas" (*pruning*) menggunakan fungsi pembatas. Umumnya, fungsi pembatas akan memangkas simpul yang *cost*-nya tidak lebih baik dari nilai *cost* terbaik saat ini atau simpul yang tidak *feasible* karena melanggar suatu *constraint*.

Pada tugas ini, algoritma *branch and bound* akan digunakan untuk menyelesaikan persoalan 15-*puzzle*. 15-puzzle. 15-*puzzle* adalah suatu teka teki dengan papan permainan berbentuk persegi yang tersusun atas 16 buah kotak (*tile*). Kotak-kotak tersebut akan diisi oleh susunan angka acak dari 1 hingga 15 dan sisanya (1 *tile*) adalah *tile* kosong. Tujuan (*objective*) dari teka teki ini adalah melakukan pergeseran pada *tile*-*tile* tersebut (dalam arah kiri, kanan, atas, dan bawah) dengan memanfaatkan *tile* kosong tadi sedemikian sehingga diperoleh susunan kotak dengan nilai terurut dari 1 hingga 15, seperti pada gambar di bawah ini:

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

Algoritma *branch and bound* yang saya implementasikan untuk menyelesaikan teka teki ini adalah sebagai berikut:

1. Pertama-tama, program utama akan melakukan pembacaan *input* matriks yang merupakan *initial state* dari *puzzle* tersebut. *Input* matriks dapat dilakukan dengan 2 cara, yaitu melalui *file* eksternal berekstensi `.txt` atau melalui pembangkit matriks yang membuat matriks *initial state* dengan susunan *random*. Setelahnya matriks tersebut akan ditampilkan ke layar.
2. Sebelum menjalankan algoritma *branch and bound*, akan dilakukan pengecekan terlebih dahulu apakah *puzzle* dengan *state* tersebut dapat diselesaikan (bisa mencapai *goal state*) dengan menggunakan fungsi `isSolvable()`. Fungsi ini akan menghitung nilai dari $\Sigma_{i=1}^{16} KURANG(i) + X$. Fungsi $KURANG(i)$ menyatakan banyaknya *tile* bernomor j sedemikian sehingga $j < i$ dan $POSISI(j) > POSISI(i)$. Sedangkan nilai X bergantung pada letak *tile* kosong pada *initial state* yang akan bernilai 1 jika berada di tile yang diarsir pada gambar di bawah ini:



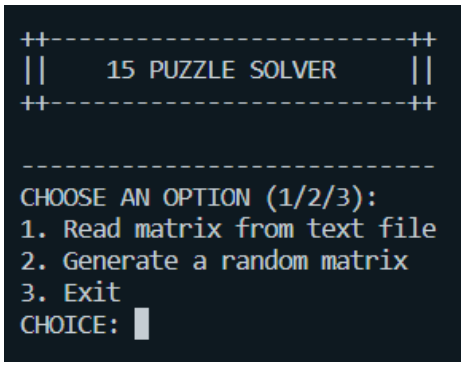
Apabila nilai $\Sigma_{i=1}^{16} KURANG(i) + X$ genap, maka *puzzle* dapat diselesaikan dan jika bernilai ganjil, *puzzle* tidak dapat diselesaikan. Kemudian semua nilai di atas juga akan ditampilkan di layar, baik untuk *puzzle* yang dapat diselesaikan, maupun yang tidak dapat diselesaikan.

3. Jika *puzzle* dapat diselesaikan (*solvable*), maka algoritma *branch and bound* yang diimplementasikan pada fungsi `solveBnB()` akan dijalankan.
   a. Pertama-tama, matriks *initial state* akan di-*construct* menjadi sebuah objek dari kelas `node` yang memiliki atribut `cost`, `idx`, `depth`, `matrix`, `direction`, `parent`. *Cost* dari suatu *node* merupakan taksiran biaya yang diperlukan untuk mencapai *goal node* dari *root node* melalui *node* tersebut. *Cost* tersebut dihitung dengan menjumlahkan panjang lintasan dari *root node* ke *node* tersebut (*depth*) dengan taksiran panjang lintasan terpendek dari *node* tersebut ke *goal node*. Taksiran tersebut diimplementasikan dalam fungsi `cellNotInPlace()` yang menghitung jumlah semua *tile*, kecuali *tile* kosong, yang saat itu tidak terletak pada tempat seharusnya (tidak sesuai dengan *goal state*).
   b. *Node* ini kemudian akan dimasukkan ke *priority queue* simpul hidup (`live_nodes`) dan ke *list* simpul yang dibangkitkan (`generated_nodes`). *Priority queue* simpul hidup akan menyimpan node secara terurut menaik (dari paling kecil ke paling besar) berdasarkan *cost*-nya.
   c. Selanjutnya akan dilakukan pengulangan selama *queue* simpul hidup tidak kosong. Dalam setiap iterasinya, akan dilakukan:
      1. *Head* dari *queue* simpul hidup akan di-*pop* sebagai simpul ekspan (`expand_node`) dan juga ditambahkan ke *list* simpul yang sudah dievaluasi (`evaluated_nodes`).
      2. Simpul ekspan saat ini akan dicek menggunakan fungsi `isSolved()`. Apabila, simpul ekspan sudah berada pada *goal state*, pencarian akan dihentikan.
      3. Kemudian, akan dibangkitkan himpunan / *list* perintah gerakan yang dapat dilakukan dari *state* tersebut menggunakan fungsi `moveCandidate()`. Gerakan yang bisa dilakukan dari suatu *state* adalah gerakan yang tidak menyebabkan *tile* bergerak keluar dari papan permainan, *tile* bergerak kembali ke posisi sebelumnya, atau *tile* kembali ke *state* yang sudah pernah dievaluasi sebelumnya.
      4. Untuk setiap arah yang dibangkitkan tersebut, akan diciptakan simpul anak yang merupakan hasil penerapan arah gerakan tersebut. Simpul-simpul anak ini akan ditambahkan ke dalam *queue* simpul hidup dan *list* simpul yang dibangkitkan.
   d. Fungsi `solveBnB()` ini akan mengembalikan *list* simpul yang dibangkitkan, *list* simpul yang sudah dievaluasi, dan simpul akhir (*goal state*).
4. Selanjutnya akan dicari lintasan dari *root node* (*state* awal) hingga ke simpul akhir (*goal state*) menggunakan fungsi `getSolutionPath()`. Fungsi ini akan mengunjungi setiap *node*, mulai dari *goal node* hingga ditemukan *node* dengan atribut `parent` yang bernilai `None`, yaitu *root node*.
5. Terakhir, akan dilakukan pencetakan lintasan solusi dari *puzzle* tersebut beserta informasi tambahan lainnya seperti waktu penyelesaian *puzzle*, jumlah *node* yang dibangkitkan, jumlah *node* yang dievaluasi, dan kedalaman *node* solusi.

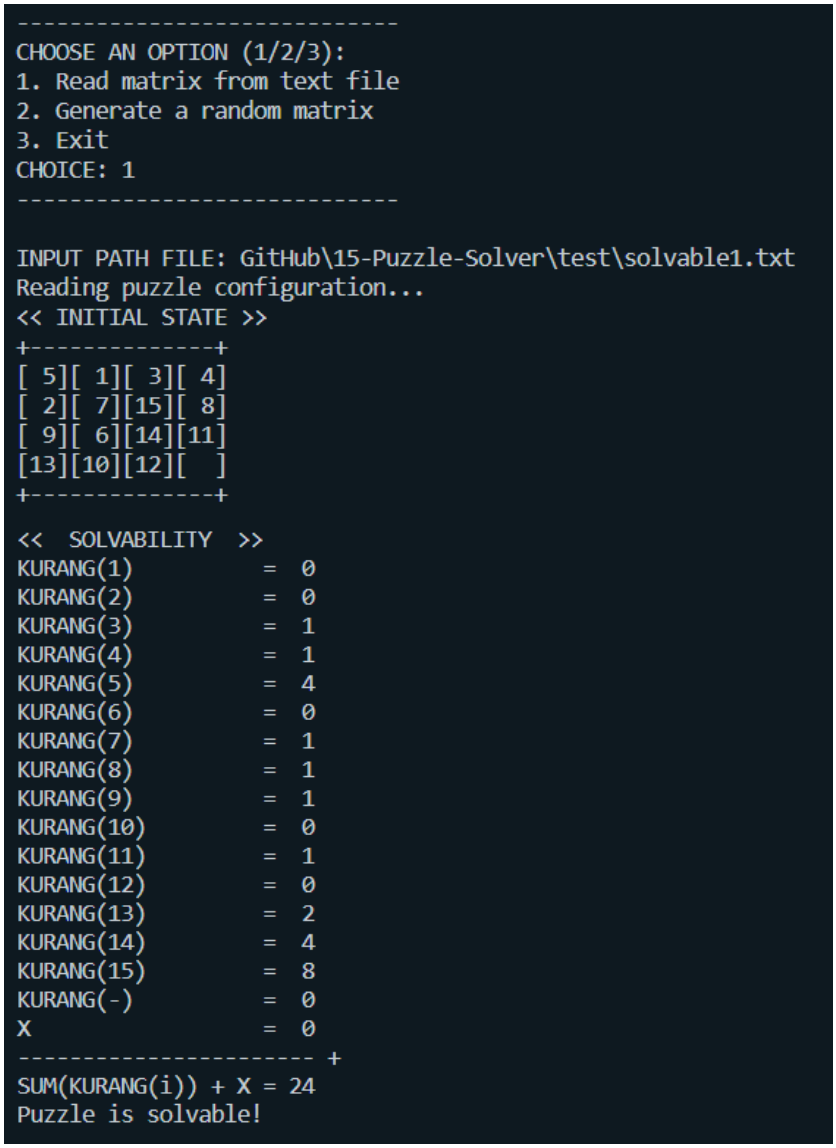
Algoritma 15-*puzzle solver* yang dibuat dapat menyelesaikan *puzzle* dengan cukup cepat untuk *puzzle* yang hanya memerlukan sekitar 20 langkah (*goal node* berada di kedalaman sekitar 20). Namun, untuk *puzzle* yang memerlukan langkah cukup banyak (>20), algoritma ini membutuhkan waktu yang cukup lama. Hal ini dikarenakan semakin besar kedalaman, maka akan semakin banyak *node* yang dibangkitkan dan dievaluasi. Namun, untuk kelima *test case* pada Bab E, semuanya dapat diselesaikan dengan baik dan benar.

B. SCREENSHOTS

1. Tampilan awal



2. *Test case* dari *file* `solvable1.txt`



```
Solving... 163

<< SOLUTION PATH >>
+------00------+
|    START     |
|  COST = 10   |
+--------------+
[ 5][ 1][ 3][ 4]
[ 2][ 7][15][ 8]
[ 9][ 6][14][11]
[13][10][12][  ]
+--------------+

+------01------+
|     LEFT     |
|  COST = 11   |
+--------------+
[ 5][ 1][ 3][ 4]
[ 2][ 7][15][ 8]
[ 9][ 6][14][11]
[13][10][  ][12]
+--------------+

+------02------+
|      UP      |
|  COST = 12   |
+--------------+
[ 5][ 1][ 3][ 4]
[ 2][ 7][15][ 8]
[ 9][ 6][  ][11]
[13][10][14][12]
+--------------+

+------03------+
|      UP      |
|  COST = 13   |
+--------------+
[ 5][ 1][ 3][ 4]
[ 2][ 7][  ][ 8]
[ 9][ 6][15][11]
[13][10][14][12]
+--------------+

+------04------+
|     LEFT     |
|  COST = 13   |
+--------------+
[ 5][ 1][ 3][ 4]
[ 2][  ][ 7][ 8]
[ 9][ 6][15][11]
[13][10][14][12]
+--------------+

+------05------+
|     LEFT     |
|  COST = 14   |
+--------------+
[ 5][ 1][ 3][ 4]
[  ][ 2][ 7][ 8]
[ 9][ 6][15][11]
[13][10][14][12]
+--------------+
```

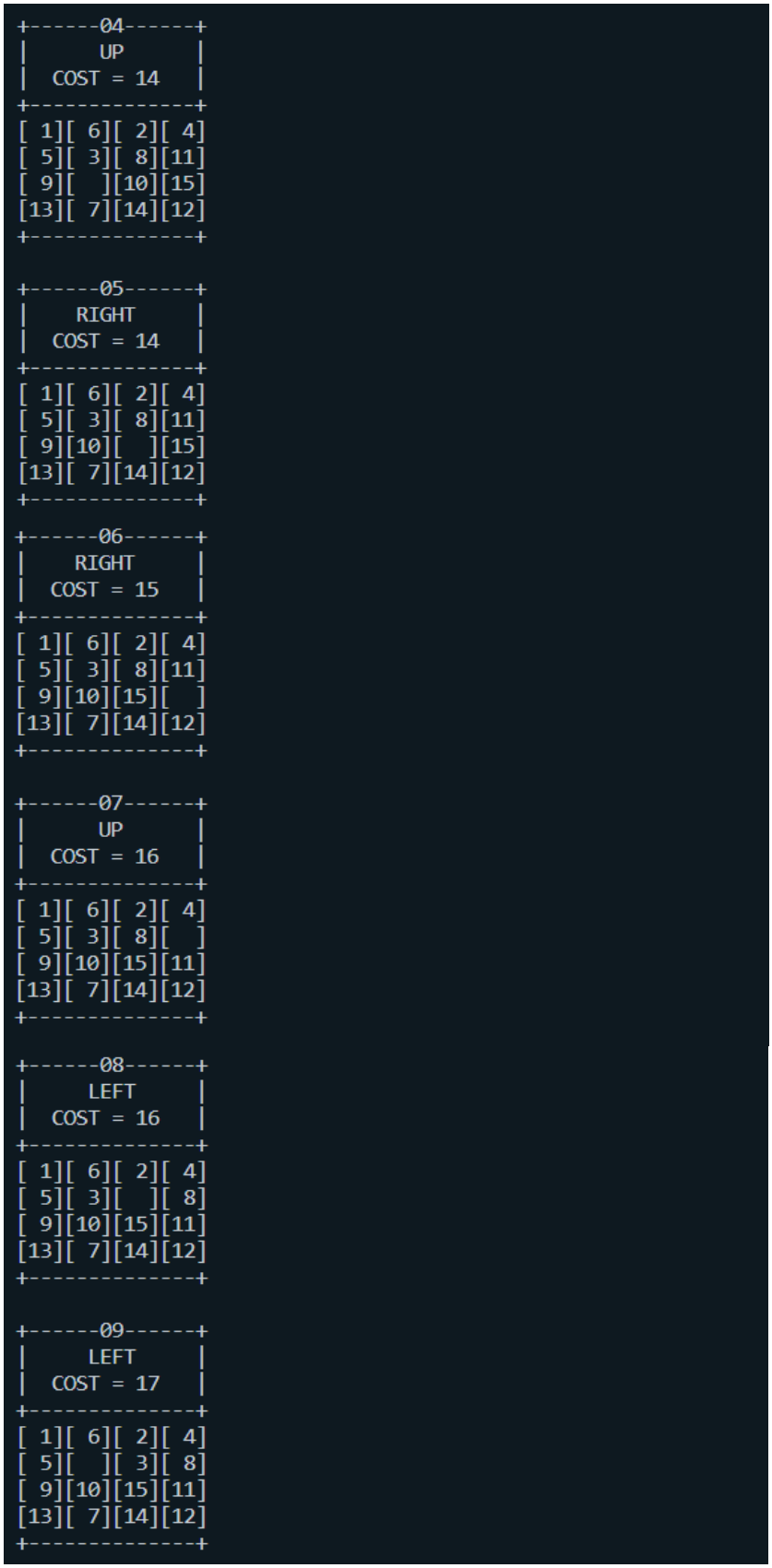
```
+------06------+
|      UP      |
|  COST = 14   |
+--------------+
[  ][ 1][ 3][ 4]
[ 5][ 2][ 7][ 8]
[ 9][ 6][15][11]
[13][10][14][12]
+--------------+

+------07------+
|    RIGHT     |
|  COST = 14   |
+--------------+
[ 1][  ][ 3][ 4]
[ 5][ 2][ 7][ 8]
[ 9][ 6][15][11]
[13][10][14][12]
+--------------+

+------08------+
|     DOWN     |
|  COST = 14   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][  ][ 7][ 8]
[ 9][ 6][15][11]
[13][10][14][12]
+--------------+

+------09------+
|     DOWN     |
|  COST = 14   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][  ][15][11]
[13][10][14][12]
+--------------+

+------10------+
|     DOWN     |
|  COST = 14   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][15][11]
[13][  ][14][12]
+--------------+

+------11------+
|    RIGHT     |
|  COST = 14   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][15][11]
[13][14][  ][12]
+--------------+
```

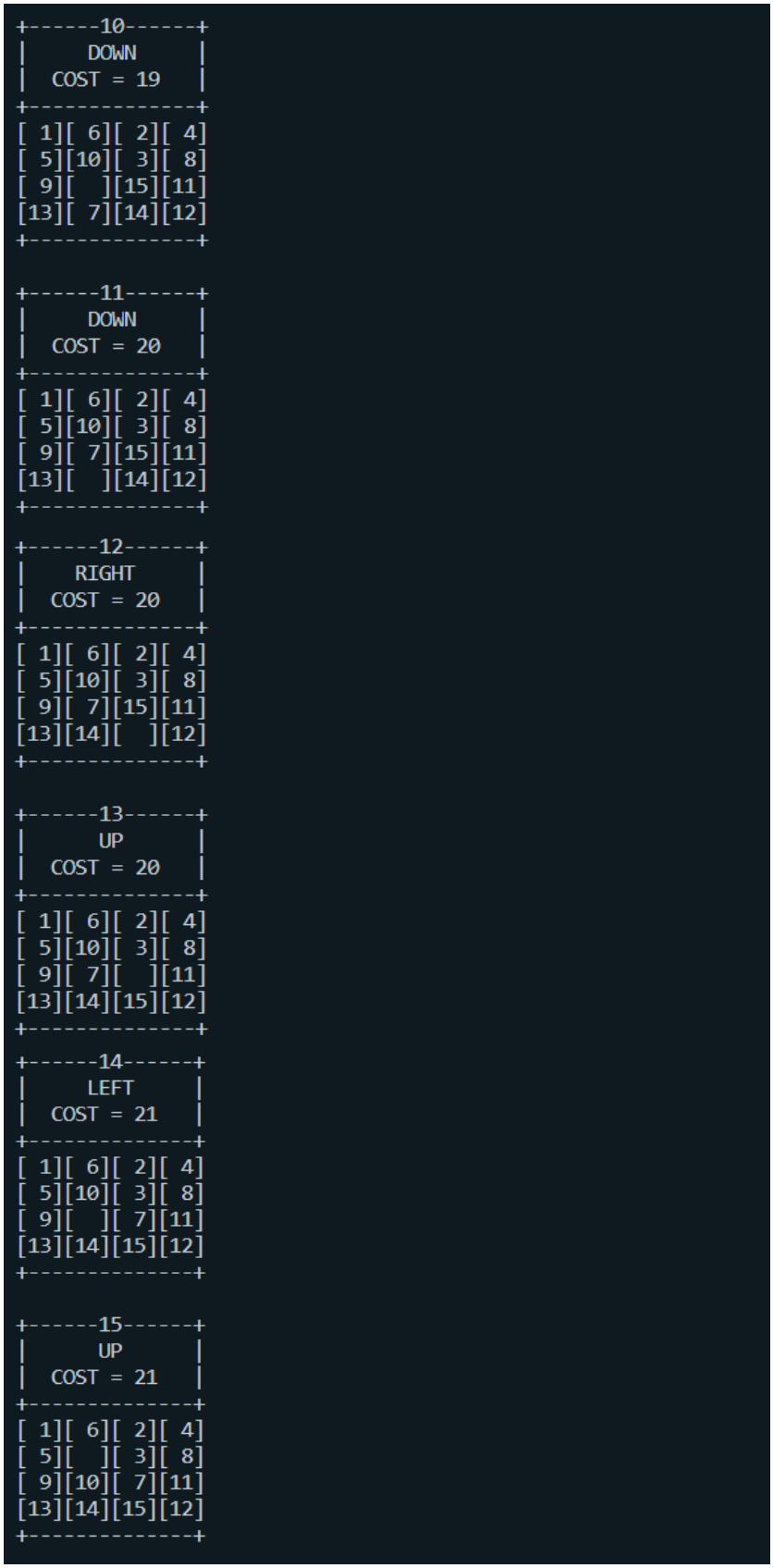
```
+------12------+
|      UP      |
|  COST = 14   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][  ][11]
[13][14][15][12]
+--------------+

+------13------+
|    RIGHT     |
|  COST = 14   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][11][  ]
[13][14][15][12]
+--------------+

+------14------+
|     DOWN     |
|  COST = 14   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][11][12]
[13][14][15][  ]
+--------------+

Puzzle succesfuly solved!
ELAPSED TIME     : 12.326 milliseconds
GENERATED NODES  : 164 nodes
EVALUATED NODES  : 74 nodes
DEPTH            : 14

Press enter to continue...
```

3. *Test case* dari *file* `solvable2.txt`

```
----------------------------
CHOOSE AN OPTION (1/2/3):
1. Read matrix from text file
2. Generate a random matrix
3. Exit
CHOICE: 1
----------------------------

INPUT PATH FILE: GitHub\15-Puzzle-Solver\test\solvable2.txt
Reading puzzle configuration...

<< INITIAL STATE >>
+--------------+
[ 1][ 6][ 2][ 4]
[ 5][ 3][  ][11]
[ 9][ 7][ 8][15]
[13][14][10][12]
+--------------+
```

```
<<  SOLVABILITY  >>
KURANG(1)          =  0
KURANG(2)          =  0
KURANG(3)          =  0
KURANG(4)          =  1
KURANG(5)          =  1
KURANG(6)          =  4
KURANG(7)          =  0
KURANG(8)          =  0
KURANG(9)          =  2
KURANG(10)         =  0
KURANG(11)         =  4
KURANG(12)         =  0
KURANG(13)         =  2
KURANG(14)         =  2
KURANG(15)         =  4
KURANG(-)          =  9
X                  =  1
---------------------- +
SUM(KURANG(i)) + X = 30
Puzzle is solvable!

Solving... 27247

<< SOLUTION PATH >>
+------00------+
|    START     |
|   COST = 9   |
+--------------+
[ 1][ 6][ 2][ 4]
[ 5][ 3][  ][11]
[ 9][ 7][ 8][15]
[13][14][10][12]
+--------------+

+------01------+
|     DOWN     |
|  COST = 10   |
+--------------+
[ 1][ 6][ 2][ 4]
[ 5][ 3][ 8][11]
[ 9][ 7][  ][15]
[13][14][10][12]
+--------------+

+------02------+
|     DOWN     |
|  COST = 11   |
+--------------+
[ 1][ 6][ 2][ 4]
[ 5][ 3][ 8][11]
[ 9][ 7][10][15]
[13][14][  ][12]
+--------------+

+------03------+
|     LEFT     |
|  COST = 13   |
+--------------+
[ 1][ 6][ 2][ 4]
[ 5][ 3][ 8][11]
[ 9][ 7][10][15]
[13][  ][14][12]
+--------------+
```

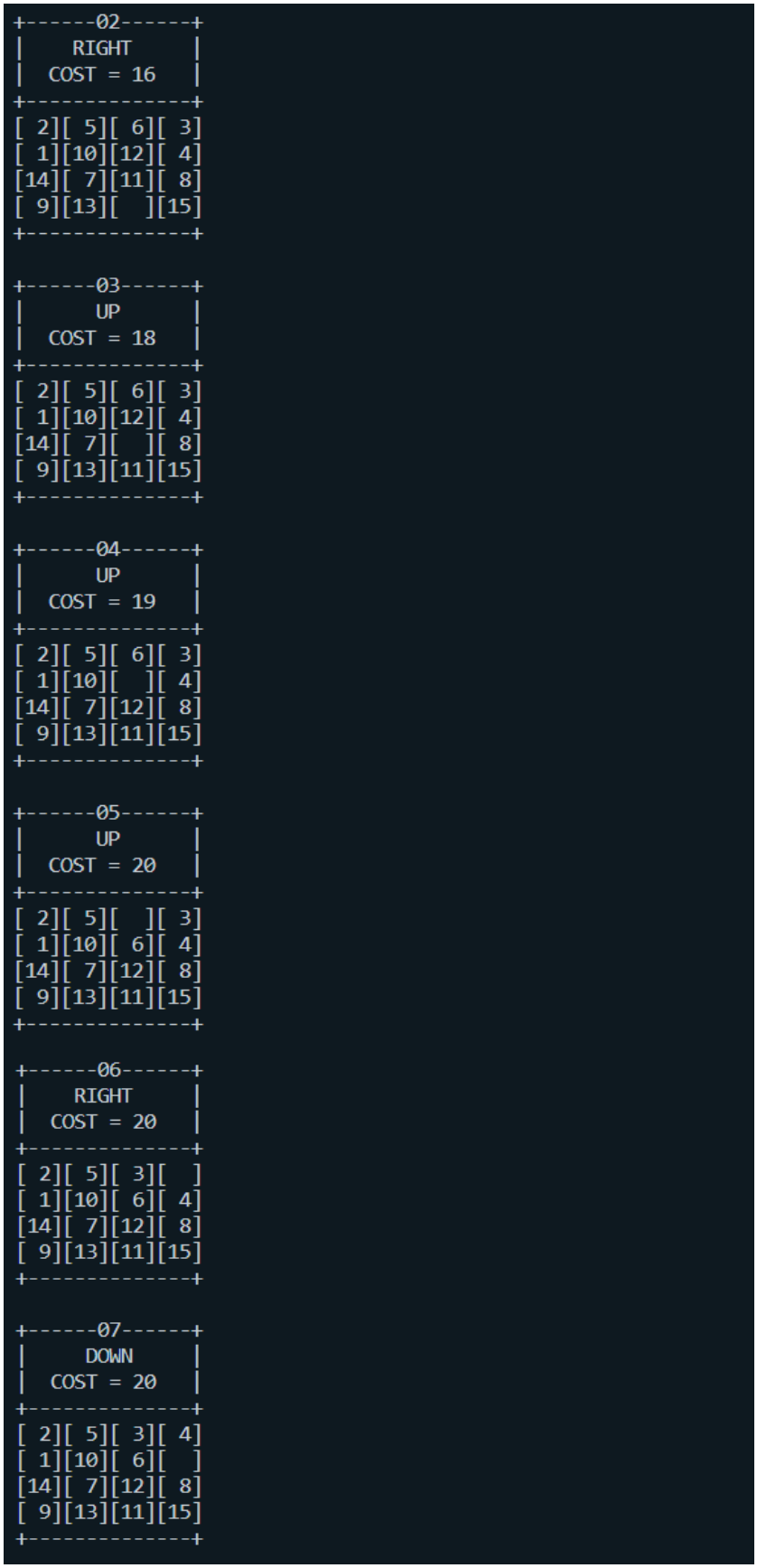

+------04------+
| UP |
| COST = 14 |
+--------------+
[1][6][2][4]
[5][3][8][11]
[9][ ][10][15]
[13][7][14][12]
+--------------+
+------05------+
| RIGHT |
| COST = 14 |
+--------------+
[1][6][2][4]
[5][3][8][11]
[9][10][ ][15]
[13][7][14][12]
+--------------+
+------06------+
| RIGHT |
| COST = 15 |
+--------------+
[1][6][2][4]
[5][3][8][11]
[9][10][15][ ]
[13][7][14][12]
+--------------+
+------07------+
| UP |
| COST = 16 |
+--------------+
[1][6][2][4]
[5][3][8][ ]
[9][10][15][11]
[13][7][14][12]
+--------------+
+------08------+
| LEFT |
| COST = 16 |
+--------------+
[1][6][2][4]
[5][3][ ][8]
[9][10][15][11]
[13][7][14][12]
+--------------+
+------09------+
| LEFT |
| COST = 17 |
+--------------+
[1][6][2][4]
[5][ ][3][8]
[9][10][15][11]
[13][7][14][12]
+--------------+


+------10------+
| DOWN |
| COST = 19 |
+--------------+
[1][6][2][4]
[5][10][3][8]
[9][ ][15][11]
[13][7][14][12]
+--------------+
+------11------+
| DOWN |
| COST = 20 |
+--------------+
[1][6][2][4]
[5][10][3][8]
[9][7][15][11]
[13][ ][14][12]
+--------------+
+------12------+
| RIGHT |
| COST = 20 |
+--------------+
[1][6][2][4]
[5][10][3][8]
[9][7][15][11]
[13][14][ ][12]
+--------------+
+------13------+
| UP |
| COST = 20 |
+--------------+
[1][6][2][4]
[5][10][3][8]
[9][7][ ][11]
[13][14][15][12]
+--------------+
+------14------+
| LEFT |
| COST = 21 |
+--------------+
[1][6][2][4]
[5][10][3][8]
[9][ ][7][11]
[13][14][15][12]
+--------------+
+------15------+
| UP |
| COST = 21 |
+--------------+
[1][6][2][4]
[5][ ][3][8]
[9][10][7][11]
[13][14][15][12]
+--------------+

```
+------16------+
|      UP      |
|  COST = 21   |
+--------------+
[ 1][  ][ 2][ 4]
[ 5][ 6][ 3][ 8]
[ 9][10][ 7][11]
[13][14][15][12]
+--------------+

+------17------+
|    RIGHT     |
|  COST = 21   |
+--------------+
[ 1][ 2][  ][ 4]
[ 5][ 6][ 3][ 8]
[ 9][10][ 7][11]
[13][14][15][12]
+--------------+

+------18------+
|     DOWN     |
|  COST = 21   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][  ][ 8]
[ 9][10][ 7][11]
[13][14][15][12]
+--------------+

+------19------+
|     DOWN     |
|  COST = 21   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][  ][11]
[13][14][15][12]
+--------------+

+------20------+
|    RIGHT     |
|  COST = 21   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][11][  ]
[13][14][15][12]
+--------------+

+------21------+
|     DOWN     |
|  COST = 21   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][11][12]
[13][14][15][  ]
+--------------+

Puzzle succesfuly solved!
ELAPSED TIME     : 7465.137 milliseconds
GENERATED NODES  : 27248 nodes
EVALUATED NODES  : 13064 nodes
DEPTH            : 21

Press enter to continue...
```

4. *Test case* dari *file* `solvable3.txt`

```
----------------------------
CHOOSE AN OPTION (1/2/3):
1. Read matrix from text file
2. Generate a random matrix
3. Exit
CHOICE: 1
----------------------------

INPUT PATH FILE: GitHub\15-Puzzle-Solver\test\solvable3.txt
Reading puzzle configuration...

<< INITIAL STATE >>
+--------------+
[ 2][ 5][ 6][ 3]
[ 1][10][12][ 4]
[14][  ][11][ 8]
[ 9][ 7][13][15]
+--------------+

<<  SOLVABILITY  >>
KURANG(1)          =  0
KURANG(2)          =  1
KURANG(3)          =  1
KURANG(4)          =  0
KURANG(5)          =  3
KURANG(6)          =  3
KURANG(7)          =  0
KURANG(8)          =  1
KURANG(9)          =  1
KURANG(10)         =  4
KURANG(11)         =  3
KURANG(12)         =  5
KURANG(13)         =  0
KURANG(14)         =  5
KURANG(15)         =  0
KURANG(-)          =  6
X                  =  1
---------------------- +
SUM(KURANG(i)) + X = 34
Puzzle is solvable!
Solving... 12877

<< SOLUTION PATH >>
+------00------+
|    START     |
|  COST = 14   |
+--------------+
[ 2][ 5][ 6][ 3]
[ 1][10][12][ 4]
[14][  ][11][ 8]
[ 9][ 7][13][15]
+--------------+

+------01------+
|     DOWN     |
|  COST = 15   |
+--------------+
[ 2][ 5][ 6][ 3]
[ 1][10][12][ 4]
[14][ 7][11][ 8]
[ 9][  ][13][15]
+--------------+
```


+------02------+
| RIGHT |
| COST = 16 |
+--------------+
[2][5][6][3]
[1][10][12][4]
[14][7][11][8]
[9][13][ ][15]
+--------------+
+------03------+
| UP |
| COST = 18 |
+--------------+
[2][5][6][3]
[1][10][12][4]
[14][7][ ][8]
[9][13][11][15]
+--------------+
+------04------+
| UP |
| COST = 19 |
+--------------+
[2][5][6][3]
[1][10][ ][4]
[14][7][12][8]
[9][13][11][15]
+--------------+
+------05------+
| UP |
| COST = 20 |
+--------------+
[2][5][ ][3]
[1][10][6][4]
[14][7][12][8]
[9][13][11][15]
+--------------+
+------06------+
| RIGHT |
| COST = 20 |
+--------------+
[2][5][3][ ]
[1][10][6][4]
[14][7][12][8]
[9][13][11][15]
+--------------+
+------07------+
| DOWN |
| COST = 20 |
+--------------+
[2][5][3][4]
[1][10][6][ ]
[14][7][12][8]
[9][13][11][15]
+--------------+

```
+------08------+
|     DOWN     |
|  COST = 20   |
+--------------+
[ 2][ 5][ 3][ 4]
[ 1][10][ 6][ 8]
[14][ 7][12][  ]
[ 9][13][11][15]
+--------------+

+------09------+
|     LEFT     |
|  COST = 20   |
+--------------+
[ 2][ 5][ 3][ 4]
[ 1][10][ 6][ 8]
[14][ 7][  ][12]
[ 9][13][11][15]
+--------------+

+------10------+
|     LEFT     |
|  COST = 21   |
+--------------+
[ 2][ 5][ 3][ 4]
[ 1][10][ 6][ 8]
[14][  ][ 7][12]
[ 9][13][11][15]
+--------------+

+------11------+
|     LEFT     |
|  COST = 22   |
+--------------+
[ 2][ 5][ 3][ 4]
[ 1][10][ 6][ 8]
[  ][14][ 7][12]
[ 9][13][11][15]
+--------------+
+------12------+
|     DOWN     |
|  COST = 22   |
+--------------+
[ 2][ 5][ 3][ 4]
[ 1][10][ 6][ 8]
[ 9][14][ 7][12]
[  ][13][11][15]
+--------------+

+------13------+
|    RIGHT     |
|  COST = 22   |
+--------------+
[ 2][ 5][ 3][ 4]
[ 1][10][ 6][ 8]
[ 9][14][ 7][12]
[13][  ][11][15]
+--------------+
```

```
+------14------+
|      UP      |
|  COST = 22   |
+--------------+
[ 2][ 5][ 3][ 4]
[ 1][10][ 6][ 8]
[ 9][  ][ 7][12]
[13][14][11][15]
+--------------+

+------15------+
|      UP      |
|  COST = 22   |
+--------------+
[ 2][ 5][ 3][ 4]
[ 1][  ][ 6][ 8]
[ 9][10][ 7][12]
[13][14][11][15]
+--------------+

+------16------+
|      UP      |
|  COST = 23   |
+--------------+
[ 2][  ][ 3][ 4]
[ 1][ 5][ 6][ 8]
[ 9][10][ 7][12]
[13][14][11][15]
+--------------+

+------17------+
|     LEFT     |
|  COST = 23   |
+--------------+
[  ][ 2][ 3][ 4]
[ 1][ 5][ 6][ 8]
[ 9][10][ 7][12]
[13][14][11][15]
+--------------+

+------18------+
|     DOWN     |
|  COST = 23   |
+--------------+
[ 1][ 2][ 3][ 4]
[  ][ 5][ 6][ 8]
[ 9][10][ 7][12]
[13][14][11][15]
+--------------+

+------19------+
|    RIGHT     |
|  COST = 23   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][  ][ 6][ 8]
[ 9][10][ 7][12]
[13][14][11][15]
+--------------+
```

```
+------20------+
|    RIGHT     |
|  COST = 23   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][  ][ 8]
[ 9][10][ 7][12]
[13][14][11][15]
+--------------+

+------21------+
|     DOWN     |
|  COST = 23   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][  ][12]
[13][14][11][15]
+--------------+

+------22------+
|     DOWN     |
|  COST = 23   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][11][12]
[13][14][  ][15]
+--------------+

+------23------+
|    RIGHT     |
|  COST = 23   |
+--------------+
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][11][12]
[13][14][15][  ]
+--------------+

Puzzle succesfuly solved!
ELAPSED TIME     : 1933.696 milliseconds
GENERATED NODES  : 12878 nodes
EVALUATED NODES  : 6395 nodes
DEPTH            : 23

Press enter to continue...
```

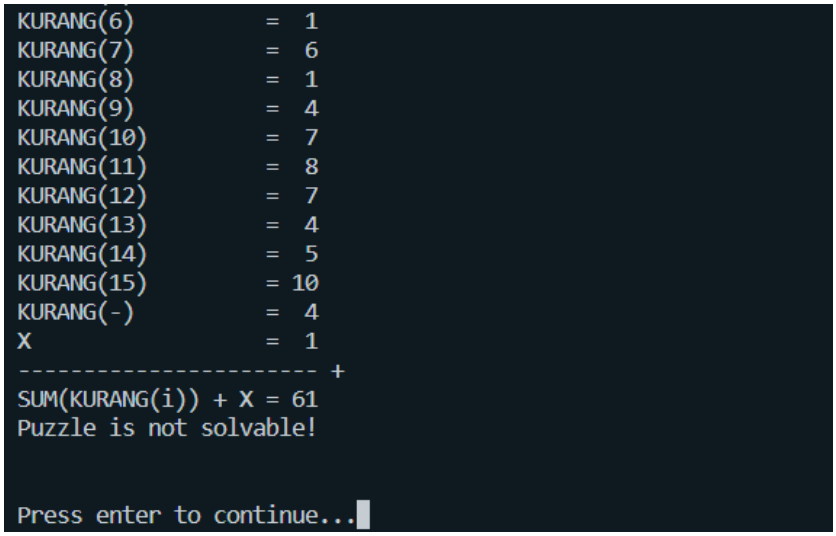
5. *Test case* dari *file* `unsolvable1.txt`

```
----------------------------
CHOOSE AN OPTION (1/2/3):
1. Read matrix from text file
2. Generate a random matrix
3. Exit
CHOICE: 1
----------------------------

INPUT PATH FILE: GitHub\15-Puzzle-Solver\test\unsolvable1.txt
Reading puzzle configuration...
```

```
<< INITIAL STATE >>
+--------------+
[15][ 7][ 4][ 8]
[11][ 6][  ][ 9]
[12][ 1][10][ 3]
[ 5][14][ 2][13]
+--------------+

<<  SOLVABILITY  >>
KURANG(1)          =  0
KURANG(2)          =  0
KURANG(3)          =  1
KURANG(4)          =  3
KURANG(5)          =  1
KURANG(6)          =  4
KURANG(7)          =  6
KURANG(8)          =  5
KURANG(9)          =  4
KURANG(10)         =  3
KURANG(11)         =  7
KURANG(12)         =  5
KURANG(13)         =  0
KURANG(14)         =  2
KURANG(15)         = 14
KURANG(-)          =  9
X                  =  1
---------------------- +
SUM(KURANG(i)) + X = 65
Puzzle is not solvable!


Press enter to continue...
```

6. *Test case* dari *file* `unsolvable2.txt`

```
----------------------------
CHOOSE AN OPTION (1/2/3):
1. Read matrix from text file
2. Generate a random matrix
3. Exit
CHOICE: 1
----------------------------

INPUT PATH FILE: GitHub\15-Puzzle-Solver\test\unsolvable2.txt
Reading puzzle configuration...

<< INITIAL STATE >>
+--------------+
[ 7][ 2][11][10]
[15][12][ 1][ 5]
[ 9][14][13][  ]
[ 3][ 6][ 8][ 4]
+--------------+

<<  SOLVABILITY  >>
KURANG(1)          =  0
KURANG(2)          =  1
KURANG(3)          =  0
KURANG(4)          =  0
KURANG(5)          =  2
```


KURANG(6) = 1
KURANG(7) = 6
KURANG(8) = 1
KURANG(9) = 4
KURANG(10) = 7
KURANG(11) = 8
KURANG(12) = 7
KURANG(13) = 4
KURANG(14) = 5
KURANG(15) = 10
KURANG(-) = 4
X = 1
---------- +
SUM(KURANG(i)) + X = 61
Puzzle is not solvable!
Press enter to continue...

C. *CHECKLIST*

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		✓

D. KODE PROGRAM

1. *File* `main.py`

File ini berisi program utama dari 15-*puzzle* solver ini seperti yang sudah dijelaskan pada Bab A di atas mengenai algorima *branch and bound*.

```
"""
Main Program for 15 Puzzle Solver (CLI)
"""

import os
import time
import util
import branch_and_bound

## HEADER
print()
print("++-------------------------++")
print("||    15 PUZZLE SOLVER     ||")
print("++-------------------------++")
print()

## MAIN PROGRAM
while(True):
    ## CHOOSE MENU
    choice = util.printMenu()

    if choice == "1":
        file = input("INPUT PATH FILE: ")
        print("Reading puzzle configuration...")
        matrix = util.fileToMatrix(file)
        if (matrix == None):
            print("File not found! Current working directory is", os.getcwd(),
".\n")
            continue
        if (not util.isMatrixValid(matrix)):
            print("Invalid matrix!\n")
            continue

    elif choice == "2":
        print("Generating a random puzzle configuration...\n")
        matrix = util.randomMatrix()

    elif choice == "3":
        print("Exiting...\n")
        break
```

```
    ## PRINT INITIAL STATE
    print("<< INITIAL STATE >>")
    util.printMatrix(matrix)

    ## CHECK SOLVABILITY
    start_time = time.time()
    print("<<  SOLVABILITY  >>")
    solvable = branch_and_bound.isSolvable(matrix)
    pause_time = time.time()

    ## PRINT SOLVABILITY INFORMATION
    util.printIsSolvableInfo(solvable[0], solvable[1], solvable[2],
solvable[3])

    ## SOLVE PUZZLE
    if (solvable[0]):
        ## SOLVING PUZZLE USING BRANCH AND BOUND
        resume_time = time.time()
        generated_nodes, evaluated_nodes, final_node =
branch_and_bound.solveBnB(matrix)
        solution_path = branch_and_bound.getSolutionPath(final_node)
        end_time = time.time()

        ## PRINT MATRIX TRANSFORMATION TO SOLVE PUZZLE
        print("<< SOLUTION PATH >>")
        for node in solution_path:
            print("+------{:02d}------+".format(node.depth))
            print("|" + (node.direction).center(14) + "|")
            print("|" + ("COST = " + str(node.cost)).center(14) + "|")
            util.printMatrix(node.matrix)
        print("Puzzle succesfuly solved!")

        ## PRINT SOLUTION INFORMATION / STATS
        elapsed_time = end_time - resume_time + pause_time - start_time
        print("ELAPSED TIME     : {} milliseconds".format(round(elapsed_time *
1000, 3)))
        print("GENERATED NODES  : {} nodes".format(len(generated_nodes)))
        print("EVALUATED NODES  : {} nodes".format(len(evaluated_nodes)))
        print("DEPTH            : {}".format(final_node.depth))

    ipt = input("\nPress enter to continue...")
    print("\n")
```

2. *File* branch_and_bound.py

File ini merupakan implementasi dari algoritma *branch and bound* itu sendiri. Fungsi utama pada *file* ini adalah solveBnB(), sedangkan fungsi lainnya merupakan fungsi tambahan yang mendukung fungsi

solveBnB() ini. Keterangan mengenai kegunaan dari setiap fungsi dapat dibaca pada bagian komentar di setiap fungsi.

```
"""
Branch and Bound Algorithm to Solve 15-Puzzle
"""

import sys
import util
import heapq


class node:
    """
    Node class for the branch and bound algorithm. Represents a tile in the 15
puzzle board.
    Attributes:
        cost        : cost of the node = sum of the cost from root to goal
node through this node
        idx         : index of the node in branch and bound tree
        depth       : depth of the node
        matrix      : the matrix of the node
        direction   : direction taken to reach this node
        parent      : parent of the node
    """
    ## CONSTRUCTOR
    def __init__(self, cost=None, idx=0, depth=None, matrix=None,
direction=None, parent=None):
        self.cost = cost
        self.idx = idx
        self.depth = depth
        self.matrix = matrix
        self.direction = direction
        self.parent = parent

    ## __lt__ (LESS THAN OPERATOR) OVERLOAD
    def __lt__(self, other):
        return self.cost < other.cost

    ## __repr__ OVERLOAD
    def __repr__(self):
        return "node(cost={}, idx={}, depth={}, matrix={},
direction={})".format(self.cost, self.idx, self.depth, self.matrix,
self.direction)

    ## __getitem__ OVERLOAD
    def __getitem__(self, key):
        return self.__dict__[key]
```

```python
def Kurang(i, matrix):
    """
    KURANG(i) = the number of tiles numbered j such that
    j < i and tile j positioned to the right (after) of tile i
    """
    # Convert matrix to array
    array = util.matrixToArray(matrix)
    sum = 0;
    # Iterate through the array and compute Kurang(i)
    for j in array:
        if j < i and array.index(j) > array.index(i):
            sum += 1
    return sum


def isSolvable(matrix):
    """
    Checks if the puzzle is solvable based on the value of
    SUM(KURANG(i))+x
    """
    # Convert matrix to array
    array = util.matrixToArray(matrix)
    # Initialize variables
    kurang = []
    x = 0
    total = 0
    # Iterate through the array and calculate SUM(KURANG(i))
    for i in array:
        kurang_i = Kurang(i, matrix)
        total += kurang_i
        heapq.heappush(kurang, (i, kurang_i))
    # Find empty cell and calculate x
    i, j = findEmptyCellIdx(matrix)
    if (i + j) % 2 != 0:
        x = 1
        total += x
    # Check if the puzzle is solvable
    if total % 2 == 0:
        return True, kurang, x, total
    else:
        return False, kurang, x, total


def isEmptyCell(x):
    """
    Checks if x is the empty cell
    """
    return x < 1 or x > 15
```

```python
def findEmptyCellIdx(matrix):
    """
    Finds the index of the empty cell in a matrix
    """
    for row in range(4):
        for col in range(4):
            if (isEmptyCell(matrix[row][col])):
                return row, col
    return None


def cellNotInPlace(matrix):
    """
    The number of tiles, except the empty cell, that are
    out of place in the current state (matrix)
    """
    # Convert matrix to array
    array = util.matrixToArray(matrix)
    count = 0
    # Iterate through the array and check if the tile is in place
    for i in range(16):
        if (not isEmptyCell(array[i])) and (array[i] != i + 1) :
            count += 1
    return count


def isSolved(matrix):
    """
    Checks if the puzzle is solved (matrix is in the goal state)
    """
    return cellNotInPlace(matrix) == 0


def cost(matrix, depth):
    """
    Calculates the cost of a node.
    c(i) = f(i) + g(i) where:
    c(i) is the total cost of the node,
    f(i) is the cost so far to reach the node from root,
    g(i) is the cost from the node to the goal node.
    """
    return cellNotInPlace(matrix)+depth


def swap(direction, matrix):
    """
    Swap two elements in a matrix based on the direction.
    Swaps the empty cell with the tile in the direction.
    """
```

```
    # Create a copy of the matrix
    temp = util.copyMatrix(matrix)
    # Find the empty cell
    i, j = findEmptyCellIdx(temp)
    if direction == "UP":
        temp[i][j], temp[i-1][j] = temp[i-1][j], temp[i][j]
    elif direction == "DOWN":
        temp[i][j], temp[i+1][j] = temp[i+1][j], temp[i][j]
    elif direction == "LEFT":
        temp[i][j], temp[i][j-1] = temp[i][j-1], temp[i][j]
    elif direction == "RIGHT":
        temp[i][j], temp[i][j+1] = temp[i][j+1], temp[i][j]
    return temp


def inverseMove(move):
    """
    Returns the inverse direction of a move
    """
    if move == "UP":
        return "DOWN"
    elif move == "DOWN":
        return "UP"
    elif move == "LEFT":
        return "RIGHT"
    elif move == "RIGHT":
        return "LEFT"


def moveCandidate(matrix, prev_move, evaluated_matrix):
    """
    Returns a list of possible moves from the current state (matrix)
    """
    # Initial move candidates
    direction = ["UP", "RIGHT", "DOWN", "LEFT"]
    # Check if the empty cell is in the edge of the board
    i, j = findEmptyCellIdx(matrix)
    if (i == 0):
        direction.remove("UP")
    elif (i == 3):
        direction.remove("DOWN")
    if (j == 0):
        direction.remove("LEFT")
    elif (j == 3):
        direction.remove("RIGHT")
    # Avoid move candidates that are the inverse of the previous move
    inv_prev_move = inverseMove(prev_move)
    if inv_prev_move in direction:
        direction.remove(inv_prev_move)
```

```
    # Avoid move candidates that produce matrix that have already been
evaluated before
    for d in direction:
        if (util.matrixToString(swap(d, matrix)) in evaluated_matrix):
            direction.remove(d)
        """for node in evaluated_nodes:
            if (swap(d, matrix) == node.matrix):
                direction.remove(d)
                break"""
    return direction


def solveBnB(matrix):
    """
    Solves the puzzle using the Branch and Bound algorithm
    """
    # Initialize variables
    generated_nodes = []
    live_nodes = []
    evaluated_nodes = []
    evaluated_states = []

    direction = "START"
    depth = 0
    idx = 0
    current_cost = cost(matrix, depth)

    # Create the root node then add it to the live nodes and generated nodes
    root = node(current_cost, idx, depth, matrix, direction)
    heapq.heappush(live_nodes, (root))
    generated_nodes.append(root)

    # Iterate until the live nodes is empty
    while (len(live_nodes) > 0):
        # Get the node with the lowest cost
        expand_node = heapq.heappop(live_nodes)
        # Add the node to the evaluated nodes
        evaluated_nodes.append(expand_node)
        evaluated_states.append(util.matrixToString(expand_node.matrix))

        # Check if the puzzle is solved
        if (isSolved(expand_node.matrix)):
            break

        # Get the possible moves from the current state
        move_candidate = moveCandidate(expand_node.matrix,
expand_node.direction, evaluated_states)
        depth = expand_node.depth + 1
```

```
        # Iterate through the possible moves
        for direction in move_candidate:
            idx += 1
            # Create a new node (children) that is the result of the move
            child_matrix = swap(direction, expand_node.matrix)
            child_cost = cost(child_matrix, depth)
            child = node(child_cost, idx, depth, child_matrix, direction,
expand_node)
            # Add the child to the live nodes and generated nodes
            heapq.heappush(live_nodes, (child))
            generated_nodes.append(child)
        print("Generating nodes... " + str(idx), end='\r')
        sys.stdout.flush()
    final_node = expand_node
    return generated_nodes, evaluated_nodes, final_node


def getSolutionPath(node):
    """
    Returns the solution path from the root node to the goal node
    """
    path = []
    while (node.parent != None):
        path.insert(0, node)
        node = node.parent
    path.insert(0, node)
    return path
```

3. *File* `util.py`

File ini berisi fungsi-fungsi utilitas yang digunakan oleh *main program* dan algoritma *branch and bound*. Fungsi-fungsi ini meliputi fungsi input/output, transformasi, validasi, dan lain-lain. Keterangan mengenai kegunaan dari setiap fungsi dapat dibaca pada bagian komentar di setiap fungsi.

```
"""
Utility Functions: Input/Output & Transformation
"""

import random
import heapq


def fileToMatrix(filePath):
    """
    Reads a file and returns a matrix of the contents
    """
    try :
        open(filePath, "r")
    except FileNotFoundError:
```

```
        return None
    with open(filePath) as f:
        matrix = []
        for line in f:
            row = []
            for i in line.split():
                if (i.isnumeric()):
                    if (int(i) >= 1 or int(i) <= 15):
                        row.append(int(i))
                else:   # empty tile in 15 puzzle problem
                    row.append(16)
            matrix.append(row)
        return matrix


def randomMatrix():
    """
    Returns a random matrix with no repeated numbers (1 to 16)
    """
    list = random.sample(range(1,17), 16)
    return arrayToMatrix(list)


def isMatrixValid(matrix):
    """
    Checks if the matrix is valid
    """
    array = matrixToArray(matrix)
    # Checks if the matrix member is unique
    unique = len(array) == len(set(array))
    # Checks the number of empty tiles
    empty_tile = sum(1 for i in array if (i < 1 or i > 15))
    return unique and empty_tile == 1


def matrixToArray(matrix):
    """
    Converts a matrix to an array
    """
    array = []
    for row in matrix:
        for col in row:
            array.append(col)
    return array


def arrayToMatrix(array):
    """
    Converts an array to a matrix
```

```
    """
    matrix = []
    for i in range(4):
        matrix.append(array[i*4:i*4+4])
    return matrix


def matrixToString(matrix):
    """
    Converts a matrix to a string
    """
    string = ""
    for row in matrix:
        for col in row:
            string += str(col) + " "
    return string


def copyMatrix(matrix):
    """
    Returns a copy of a matrix
    """
    copy = []
    for row in matrix:
        copy.append(row.copy())
    return copy


def findMatrixIndex(matrix, value):
    """
    Finds the index of a value in a matrix
    """
    for row in range(4):
        for col in range(4):
            if matrix[row][col] == value:
                return row, col
    # value not found
    return None


def printMatrix(matrix):
    """
    Prints a matrix as a 15 puzzle board
    """
    print("+--------------+")
    for row in matrix:
        for col in row:
            if (col >= 1 and col <= 15):
                print("[" + str(col).rjust(2, ' '), end=']')
```

```
            else: # empty tile in 15 puzzle problem
                print("[  ]", end="")
        print()
    print("+--------------+\n")


def printMenu():
    """
    Prints the menu for the main program
    """
    while (True):
        print("------------------------------")
        print("CHOOSE AN OPTION (1/2/3):")
        print("1. Read matrix from text file")
        print("2. Generate a random matrix")
        print("3. Exit")
        choice = input("CHOICE: ")
        print("------------------------------")
        if choice == "1" or choice == "2" or choice == "3":
            break
        print("Invalid choice, please insert 1/2/3!\n")
    print()
    return choice


def printIsSolvableInfo(is_solvable, kurang, x, total):
    """
    Prints the information about the solvability of a matrix
    """
    # Prints the value of Kurang(i) for each tile i
    while (len(kurang) > 0):
        temp = heapq.heappop(kurang)
        print("KURANG({})".format("-" if (temp[0]<1 or temp[0]>15) else
temp[0]).ljust(19) + "= " + str(temp[1]).rjust(2))
    # Prints the x value indicating where the empty tile is
    print("X                   =  " + str(x))
    print("-"*23,"+")
    # Prints the value of SUM(Kurang(i))+x
    print("SUM(KURANG(i)) + X = {}".format(total))
    # Prints the solvability of the matrix
    if is_solvable:
        print("Puzzle is solvable!\n")
    else:
        print("Puzzle is not solvable!\n")
```

E. *TEST CASE*

Data uji (*test case*) yang digunakan terdiri dari 2 jenis kasus, yaitu *puzzle* yang dapat diselesaikan (*solvable*) dan *puzzle* yang tidak dapat diselesaikan (*unsolvable*). Berikut ini adalah kelima test case tersebut:

1. *File* `solvable1.txt`
   5 1 3 4
   2 7 15 8
   9 6 14 11
   13 10 12 -

2. *File* `solvable2.txt`
   1 6 2 4
   5 3 - 11
   9 7 8 15
   13 14 10 12

3. *File* `solvable3.txt`
   2 5 6 3
   1 10 12 4
   14 - 11 8
   9 7 13 15

4. *File* `unsolvable1.txt`
   15 7 4 8
   11 6 - 9
   12 1 10 3
   5 14 2 13

5. File `unsolvable2.txt`
   7 2 11 10
   15 12 1 5
   9 14 13 -
   3 6 8 4

F. TAUTAN KODE PROGRAM

Kode program beserta kelima *test case* di atas dapat diakses pada *link* GitHub berikut ini:
https://github.com/HanselTanoto/Tucil3_13520046