

STRATEGI ALGORITMA
15 PUZZLE SOLVER DENGAN ALGORITMA *BRANCH AND BOUND*
LAPORAN TUGAS KECIL 3

Diajukan sebagai Salah Satu Tugas Kecil Mata Kuliah Strategi Algoritma pada Semester 4



Azka Syauqy Irsyad

13520107

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2022

I. Algoritma *Branch and Bound* pada Pengaplikasian 15 *Puzzle Solver*

Algoritma *branch and bound* adalah suatu algoritma yang dapat digunakan dalam suatu permasalahan optimisasi. Pada penerapannya, algoritma ini memanfaatkan pohon ruang status yang digunakan untuk mencari solusi terbaik yang memungkinkan. Proses pencarian akan mengevaluasi kemungkinan pengambilan simpul selanjutnya yang memberikan solusi paling baik di antara simpul lainnya, terus hingga *goal state* tercapai. Penerapan algoritma ini salah satunya adalah dalam membuat 15 *puzzle solver*.

Persoalan 15 *puzzle* adalah suatu persoalan dimana terdapat suatu *puzzle* yang digambarkan sebagai matriks. Elemen-elemen dalam *puzzle* tersebut merupakan angka 1 s.d. 15 yang disertai dengan elemen kosong yang urutannya diacak. Pencarian solusi dari persoalan ini adalah menyusun elemen-elemen tersebut dari terkecil hingga terbesar sesuai urutan secara horizontal, dimana elemen kosong tersebut berada di urutan paling akhir. Pergeseran elemen pun hanya boleh yang merupakan elemen kosong, dimana pergeseran dapat dilakukan ke atas, bawah, kiri, atau kanan.

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

Gambar 1 Susunan Akhir 15 *Puzzle*

Untuk menyelesaikan persoalan ini, perlu dihitung terlebih dahulu apakah status awal dari *puzzle* dapat dicapai, dengan menggunakan teorema $\sum_{i=1}^{16} Kurang(i) + X$ harus menghasilkan nilai genap, dimana nilai X adalah 1 jika posisi elemen kosong berada ketika jumlah baris dan kolom adalah ganjil dan bernilai 0 jika sebaliknya. *Kurang(i)* adalah

banyaknya elemen yang bernilai lebih kecil dari i namun posisinya lebih besar darinya. Jika teorema di atas terpenuhi, maka pencarian solusi dapat dilakukan, tidak sebaliknya.

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

$$\sum_{i=1}^{16} Kurang(i) + X = 37 + 0 = 37$$

Gambar 2 Contoh *Puzzle* yang Tidak Bisa Diselesaikan

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

$$\sum_{i=1}^{16} Kurang(i) + X = 15 + 1 = 16$$

Gambar 3 Contoh *Puzzle* yang Bisa Diselesaikan

Alur kerja program yang dibuat dimulai dengan memilih *mode* penyusunan dari *puzzle*, yaitu dibangkitkan secara *random* oleh program atau membaca dari file. Jika mode yang dipilih adalah *random*, maka *puzzle* akan dibuat secara random urutan-urutan elemennya. Jika dari input file, maka program akan membaca matriks dari file masukan dengan ekstensi yang berlaku adalah .txt. Setelah kedua matriks *puzzle* sudah tersedia, akan ditampilkan bentuk *puzzle* yang akan dicari solusinya.

Setelah itu, program akan melakukan perhitungan nilai $Kurang(i)$ dari setiap elemen dari matriks tersebut. Program juga akan mencetak hasil perhitungan tersebut ke layar, dimana elemen 16 menandakan elemen 'kosong' seperti pada *template* persoalan. Program

juga akan mencetak hasil perhitungan dari $\sum_{i=1}^{16} Kurang(i) + X$, yang kemudian dievaluasi

apakah dari nilai tersebut status tujuan dapat dicapai atau tidak. Jika status tujuan tidak dapat dicapai, program akan mengeluarkan pesan bahwa persoalan tidak dapat menghasilkan solusi. Jika berhasil, program akan mulai menghitung waktu eksekusi dan pencarian solusi pun dimulai.

Dalam pencarian solusinya, dibuat suatu *priority queue* yang di dalamnya berisi simpul-simpul dari pergerakan matriks. Diambil nilai *cost* berupa jumlah elemen tidak kosong yang urutannya tidak sesuai dengan *goal state*. Matriks yang telah dibaca tadi akan menjadi akar tersebut dan dimasukkan ke dalam *priority queue* tersebut. Lalu, program akan melakukan *pop* untuk membaca tiap simpul yang ada di *priority queue* terus-menerus selama *priority queue* tidak kosong dan waktu eksekusi belum melebihi limit. Dalam proses *loopin* tersebut, akan dicek daftar *move* yang valid, yaitu dicek posisi dan *move* yang sedang diambil. Daftar *move* tersebut akan diiterasi untuk pengecekan simpul berikutnya setelah dibuat menjadi simpul anak dan di *push* ke dalam *priority queue*. Perlu diingat juga bahwa dalam struktur data ini, *push* elemen didasarkan pada *value cost* yang ditambahkan dengan kedalaman simpul tersebut, baru kemudian urutan masuk.

Proses pencarian akan terus berlangsung hingga ada kondisi yang membuat pencarian dihentikan selain kondisi yang disebutkan di atas tadi, dimana kondisi tersebut belum menunjukkan ditemukannya solusi. Pencarian yang menghasilkan solusi ini ditandai ketika nilai *cost* adalah nol, yang menandakan bahwa matriks awal sudah merupakan solusi, atau nilai *cost* merupakan nilai kedalaman dari simpul tersebut. Ketika kondisi itu tercapai, waktu eksekusi pencarian akan dihentikan dan program mulai mencetak langkah matriks pencarian dari awal hingga matriks tujuan sesuai dengan urutan *move* yang diambil. Program juga akan mencetak jumlah simpul yang dibangkitkan dalam pencarian solusi dan lama waktu eksekusi pencarian.

Perlu diketahui juga bahwa matriks *puzzle* yang dibangun dengan *random* oleh program, dan beberapa contoh kasus lain serupa yang dapat terjadi juga pada masukan dari file, dapat menghasilkan *looping* yang amat panjang, yang membuat pemakaian memori yang berlebihan. Oleh karena itu, pencarian solusi dapat membuat tidak berhasilnya menunjukkan solusi pencarian dikarenakan program masih mencari melebihi *time limit* yang ditentukan. Jika persoalan mengalami hal ini, akan dikeluarkan pesan bahwa waktu persoalan terlalu lama dieksekusi sehingga tidak dapat dilanjutkan.

II. Screenshot Input dan Output Program

Berikut adalah input dan output dari pembacaan *puzzle* dari file.



Gambar 4 Input *matrix1.txt*

15 Puzzle Solver with Branch and Bound Algorithm

Daftar Pilihan Mode :

1. Random
2. Input File

Pilih mode : 2

Masukkan nama file puzzle yang ingin dicari solusinya: matrix1
Puzzle yang akan dicari adalah seperti berikut:

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

Nilai Kurang(i) untuk setiap ubin adalah sebagai berikut:

1	:	0
2	:	0
3	:	0
4	:	0
5	:	0
6	:	0
7	:	0
8	:	1
9	:	1
10	:	1
11	:	0
12	:	0
13	:	1
14	:	1
15	:	1
16	:	9

Nilai dari Sigma Kurang(i) = 16

Persoalan ini dapat diselesaikan dengan langkah sebagai berikut:

Step taken: Root

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

Step taken: Move Down

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

Step taken: Move Right

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

Step taken: Move Down

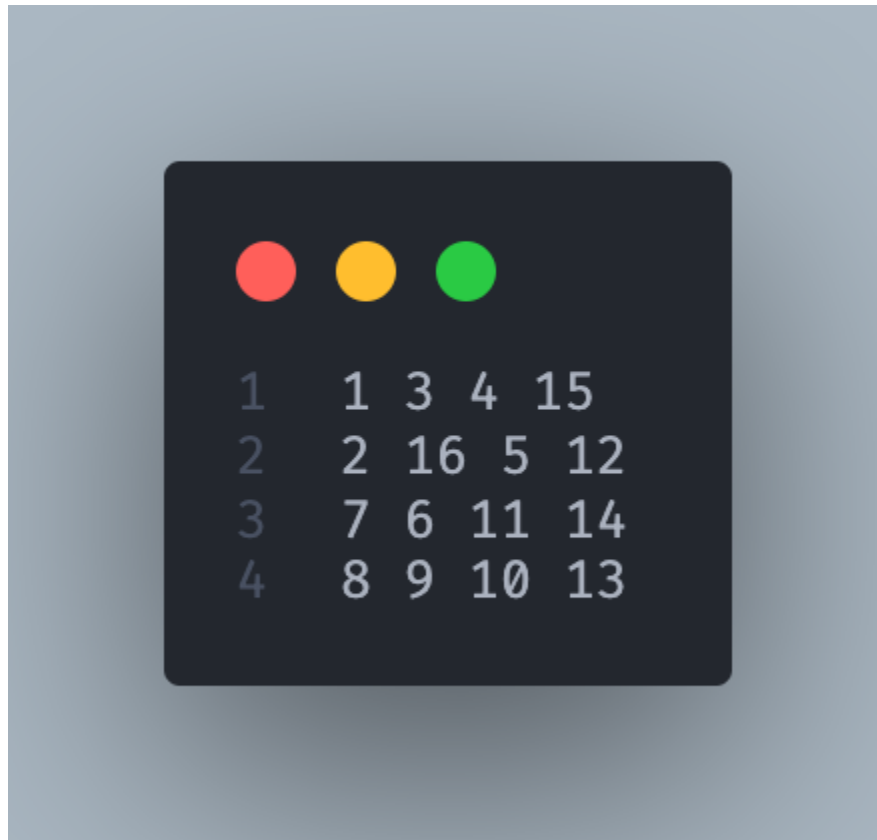
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	-

Finished!

Simpul dibangkitkan = 9

Time execution = 0.0004017353057861328 sekon

Gambar 5 Hasil Output *matrix1.txt*



Gambar 6 Input *matrix2.txt*

```

15 Puzzle Solver with Branch and Bound Algorithm

Daftar Pilihan Mode :
1. Random
2. Input File

Pilih mode : 2

Masukkan nama file puzzle yang ingin dicari solusinya: matrix2
Puzzle yang akan dicari adalah seperti berikut:

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

Nilai Kurang(i) untuk setiap ubin adalah sebagai berikut:

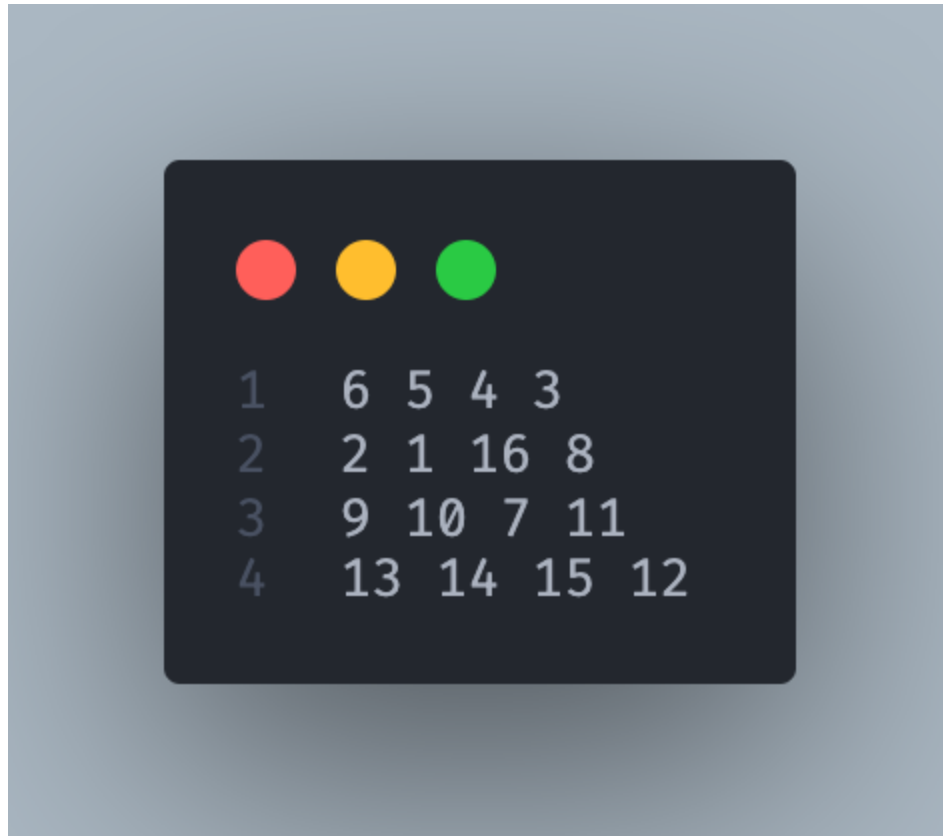
1      :      0
2      :      0
3      :      1
4      :      1
5      :      0
6      :      0
7      :      1
8      :      0
9      :      0
10     :      0
11     :      3
12     :      6
13     :      0
14     :      4
15     :      11
16     :      10

Nilai dari Sigma Kurang(i)      =      37

Persoalan ini tidak dapat menghasilkan solusi

```

Gambar 7 Hasil Output *matrix2.txt*



Gambar 7 Input *matrix3.txt*

```

15 Puzzle Solver with Branch and Bound Algorithm

Daftar Pilihan Mode :
1. Random
2. Input File

Pilih mode : 2

Masukkan nama file puzzle yang ingin dicari solusinya: matrix3
Puzzle yang akan dicari adalah seperti berikut:

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

Nilai Kurang(i) untuk setiap ubin adalah sebagai berikut:

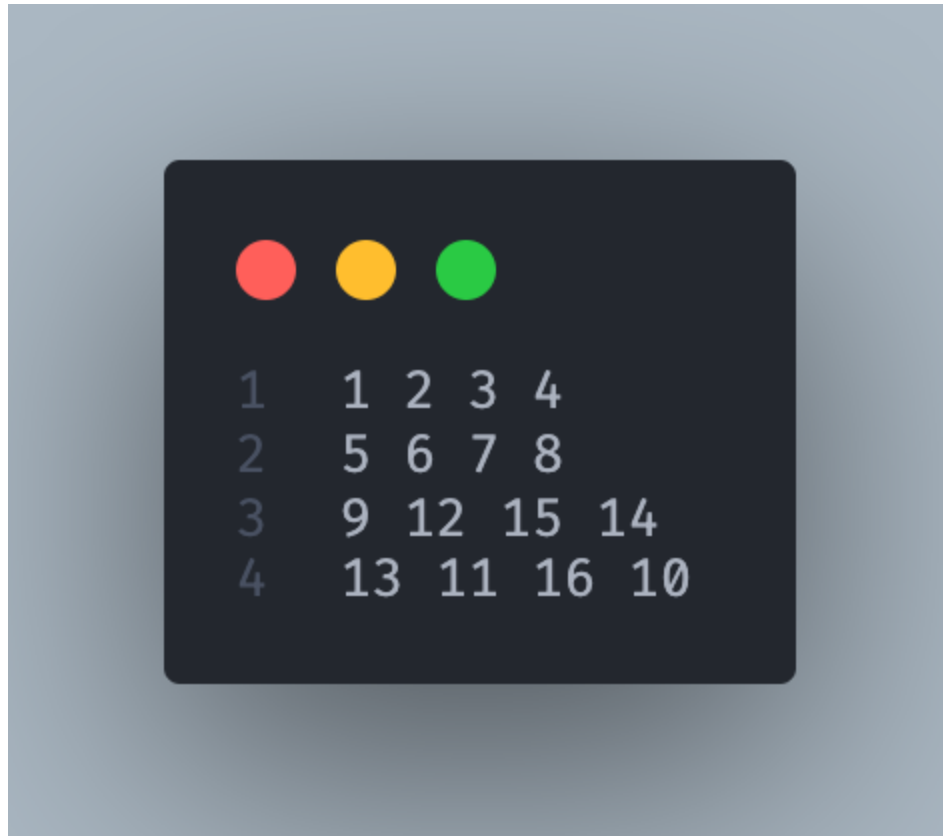
1      :      0
2      :      1
3      :      2
4      :      3
5      :      4
6      :      5
7      :      0
8      :      1
9      :      1
10     :      1
11     :      0
12     :      0
13     :      1
14     :      1
15     :      1
16     :      9

Nilai dari Sigma Kurang(i)      =      31

Persoalan ini tidak dapat menghasilkan solusi

```

Gambar 8 Hasil Output *matrix3.txt*



Gambar 9 Input *matrix4.txt*

15 Puzzle Solver with Branch and Bound Algorithm

Daftar Pilihan Mode :

1. Random
2. Input File

Pilih mode : 2

Masukkan nama file puzzle yang ingin dicari solusinya: matrix4

Puzzle yang akan dicari adalah seperti berikut:

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

Nilai Kurang(i) untuk setiap ubin adalah sebagai berikut:

1	:	0
2	:	0
3	:	0
4	:	0
5	:	0
6	:	0
7	:	0
8	:	0
9	:	0
10	:	0
11	:	1
12	:	2
13	:	2
14	:	3
15	:	4
16	:	1

Nilai dari Sigma Kurang(i) = 14

Persoalan ini dapat diselesaikan dengan langkah sebagai berikut:

Step taken: Root

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

Step taken: Move Right

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

Step taken: Move Up

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

Step taken: Move Left

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

Step taken: Move Left

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

Step taken: Move Down

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

Step taken: Move Right

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

Step taken: Move Right

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

Step taken: Move Up

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

Step taken: Move Left

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

Step taken: Move Left

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

Step taken: Move Down

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

Step taken: Move Right

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

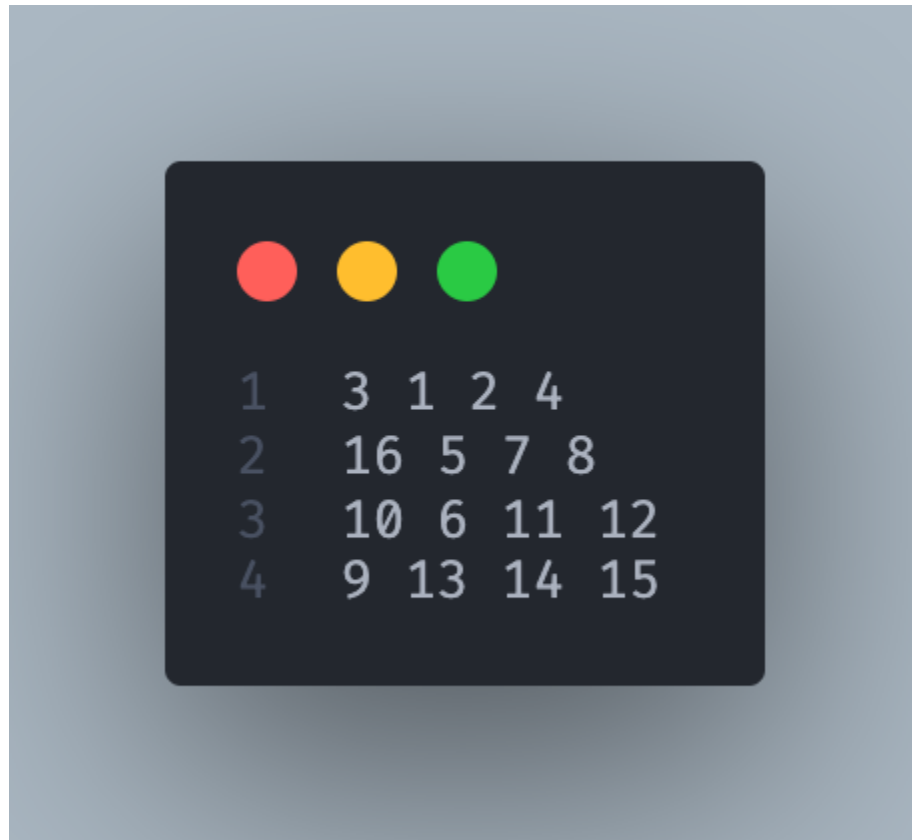
Step taken: Move Right

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

Finished!

Simpul dibangkitkan	=	632
Time execution	=	0.01398921012878418 sekon

Gambar 10 dan 11 Hasil Output *matrix4.txt*



Gambar 12 Input *matrix5.txt*

15 Puzzle Solver with Branch and Bound Algorithm

Daftar Pilihan Mode :

1. Random
2. Input File

Pilih mode : 2

Masukkan nama file puzzle yang ingin dicari solusinya: matrix5
Puzzle yang akan dicari adalah seperti berikut:

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

Nilai Kurang(i) untuk setiap ubin adalah sebagai berikut:

1	:	0
2	:	0
3	:	2
4	:	0
5	:	0
6	:	0
7	:	1
8	:	1
9	:	0
10	:	2
11	:	1
12	:	1
13	:	0
14	:	0
15	:	0
16	:	11

Nilai dari Sigma Kurang(i) = 20

Persoalan ini dapat diselesaikan dengan langkah sebagai berikut:

Step taken: Root

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

Step taken: Move Right

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

Step taken: Move Down

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

Step taken: Move Left

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

Step taken: Move Up

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

Step taken: Move Up

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

Step taken: Move Right

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

Step taken: Move Down

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

Step taken: Move Left

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

Step taken: Move Down

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

Step taken: Move Down

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

Step taken: Move Right

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

Step taken: Move Right

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

Step taken: Move Up

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

Step taken: Move Up

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


```

Step taken: Move Left
1      6      2      4
5      -      3      8
9      10     7      12
13     14     11     15

Step taken: Move Up
1      -      2      4
5      6      3      8
9      10     7      12
13     14     11     15

Step taken: Move Right
1      2      -      4
5      6      3      8
9      10     7      12
13     14     11     15

Step taken: Move Down
1      2      3      4
5      6      -      8
9      10     7      12
13     14     11     15

Step taken: Move Down
1      2      3      4
5      6      7      8
9      10     -      12
13     14     11     15

Step taken: Move Down
1      2      3      4
5      6      7      8
9      10     11     12
13     14     -      15

Step taken: Move Right
1      2      3      4
5      6      7      8
9      10     11     12
13     14     15     -

Finished!

Simpul dibangkitkan      =      15539
Time execution           =      0.1746060848236084 sekon

```

Gambar 13, 14, dan 15 Hasil Output *matrix5.txt*


III. Kode Program

Berikut adalah kode program dalam *fifteenPuzzle.py*



```
1  # class PriorityQueue
2  class PrioQueue:
3      # Constructor
4      def __init__(self):
5          self.heap = []
6
7      # Method push based on value
8      def push(self, value):
9          heappush(self.heap, value)
10
11     # Method pop minimum element
12     def pop(self):
13         return heappop(self.heap)
14
15     # Method isEmpty
16     def isEmpty(self):
17         if not self.heap:
18             return True
19         return False
```

Gambar 16 Kelas *PrioQueue*




```

1  # class Node
2  class Node:
3      # Constructor
4      def __init__(self, root, matrix, emptyPosRow, emptyPosCol, diff, depth, move):
5          self.root = root
6          self.matrix = matrix
7          self.emptyPosRow = emptyPosRow
8          self.emptyPosCol = emptyPosCol
9          self.diff = diff
10         self.depth = depth
11         self.move = move
12
13     # Method agar prioQueue terbentuk berdasarkan nilai diff
14     def __lt__(self, other):
15         return self.diff < other.diff

```

Gambar 17 Kelas *Node*



```

1  # fungsi mencari indeks elemen
2  def findLocElmt(num, inMatrix):
3      for row in range(4):
4          for col in range(4):
5              if inMatrix[row][col] == num:
6                  return row, col

```

Gambar 18 Fungsi Mencari Indeks Elemen



```
1  def kurangI(num, inMatrix):
2      row, col = findLocElmt(num, inMatrix)
3      count = 0
4      if col != 3:
5          for i in range(col, 3):
6              if inMatrix[row][i+1] < num:
7                  count += 1
8      if row != 3:
9          for i in range(row + 1, 4):
10             for j in range(4):
11                 if inMatrix[i][j] < num:
12                     count += 1
13     return count
```

Gambar 19 Fungsi Mencari Nilai *Kurang(i)*

```

1  # fungsi untuk mengecek apakah matriks dapat diselesaikan
2  def isSolveable(inMatrix):
3      global row16, col16, solveable
4      row16, col16 = findLocElmt(16, inMatrix)
5      if (row16 + col16) % 2 == 1:
6          sigma = 1 # nilai X posisi yang diarsir
7      else:
8          sigma = 0 # nilai X posisi yang tidak diarsir
9      for i in range(4):
10         for j in range(4):
11             # print nilai kurangI dari elemen
12             print(compareMatrix[i][j], "\t:\t", kurangI(compareMatrix[i][j], inMatrix), end="\n")
13             # tambahkan ke dalam nilai sigma
14             sigma += kurangI(inMatrix[i][j], inMatrix)
15     print("\nNilai dari Sigma Kurang(i)\t=\t", sigma, end="\n")
16     if sigma % 2 == 0:
17         solveable = True

```

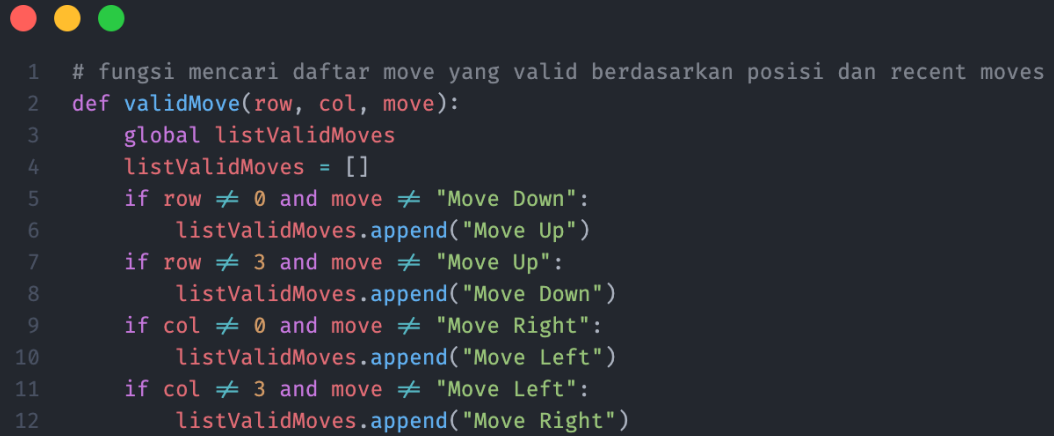
Gambar 20 Fungsi *isSolveable*

```

1  # fungsi mencari jumlah elemen yang berbeda terhadap compareMatrix
2  def getDiff(inMatrix):
3      global countNodes
4
5      diff = 0
6      for i in range(4):
7          for j in range(4):
8              if inMatrix[i][j] != compareMatrix[i][j] and inMatrix[i][j] != 16:
9                  diff += 1
10         countNodes += 1 # menambah jumlah node yang dibangkitkan
11     return diff

```

Gambar 21 Fungsi Mencari Nilai *Cost*



```
1 # fungsi mencari daftar move yang valid berdasarkan posisi dan recent moves
2 def validMove(row, col, move):
3     global listValidMoves
4     listValidMoves = []
5     if row != 0 and move != "Move Down":
6         listValidMoves.append("Move Up")
7     if row != 3 and move != "Move Up":
8         listValidMoves.append("Move Down")
9     if col != 0 and move != "Move Right":
10        listValidMoves.append("Move Left")
11    if col != 3 and move != "Move Left":
12        listValidMoves.append("Move Right")
```

Gambar 22 Fungsi Mencari *ValidMove*



```
1  # fungsi bergerak ke atas
2  def moveUp(row, col):
3      return row - 1, col
4
5  # fungsi bergerak ke bawah
6  def moveDown(row, col):
7      return row + 1, col
8
9  # fungsi bergerak ke kiri
10 def moveLeft(row, col):
11     return row, col - 1
12
13 # fungsi bergerak ke kanan
14 def moveRight(row, col):
15     return row, col + 1
```

Gambar 23 Daftar Fungsi *Move*

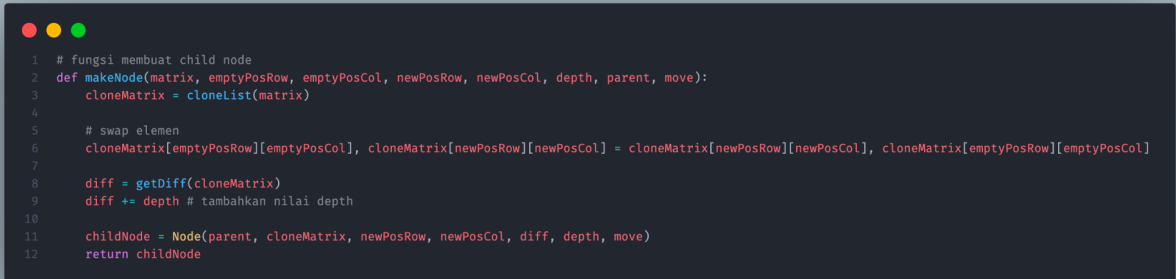


```

1  # fungsi copy matrix
2  def cloneList(li):
3      newList = [[0 for j in range(4)] for i in range(4)]
4      for i in range(4):
5          for j in range(4):
6              newList[i][j] = li[i][j]
7      return newList

```

Gambar 24 Fungsi untuk Menyalin Matriks

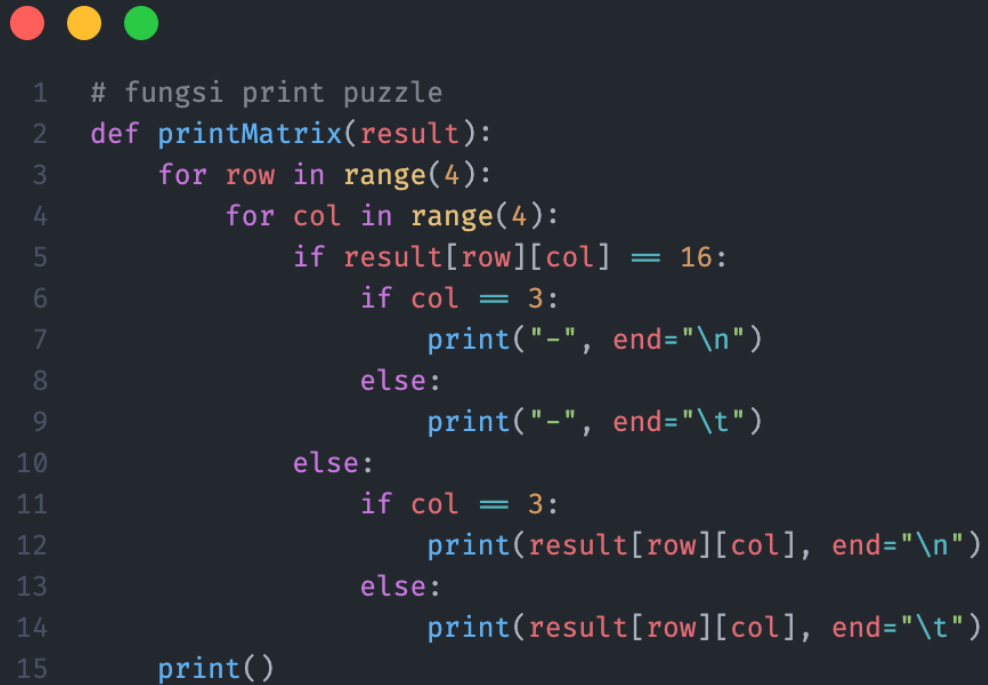


```

1  # fungsi membuat child node
2  def makeNode(matrix, emptyPosRow, emptyPosCol, newPosRow, newPosCol, depth, parent, move):
3      cloneMatrix = cloneList(matrix)
4
5      # swap elemen
6      cloneMatrix[emptyPosRow][emptyPosCol], cloneMatrix[newPosRow][newPosCol] = cloneMatrix[newPosRow][newPosCol], cloneMatrix[emptyPosRow][emptyPosCol]
7
8      diff = getDiff(cloneMatrix)
9      diff += depth # tambahkan nilai depth
10
11     childNode = Node(parent, cloneMatrix, newPosRow, newPosCol, diff, depth, move)
12     return childNode

```

Gambar 25 Fungsi Membuat *Child Node*



```
1 # fungsi print puzzle
2 def printMatrix(result):
3     for row in range(4):
4         for col in range(4):
5             if result[row][col] == 16:
6                 if col == 3:
7                     print("-", end="\n")
8                 else:
9                     print("-", end="\t")
10            else:
11                if col == 3:
12                    print(result[row][col], end="\n")
13                else:
14                    print(result[row][col], end="\t")
15        print()
```

Gambar 26 Fungsi Mencetak *Puzzle*



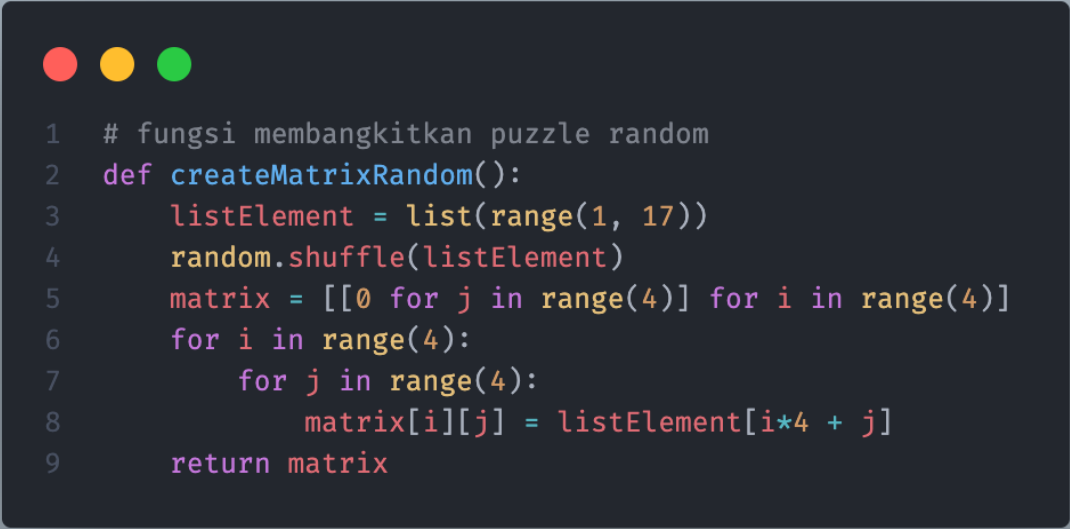
```
1 # fungsi print rute dari root ke ditemukannya solusi
2 def printRoutes(root):
3     if root == None:
4         return
5
6     printRoutes(root.root)
7     print("Step taken: " + root.move)
8     printMatrix(root.matrix)
9     print()
```

Gambar 27 Fungsi Mencetak Matriks dari *Root* ke Solusi



```
1 # fungsi untuk mencari solusi
2 def findSolution(inMatrix):
3     global listValidMoves, countNodes, stop, getSolution
4     # membuat PrioQueue
5     liveNode = PrioQueue()
6
7     diff = getDiff(inMatrix)
8
9     # membuat root node
10    root = Node(None, inMatrix, row16, col16, diff, 0, "Root")
11
12    # menambahkan root node ke dalam PrioQueue
13    liveNode.push(root)
14
15    # looping untuk mencari solusi
16    while not liveNode.isEmpty() and time.time() - start < 180:
17        getNode = liveNode.pop()
18        if getNode.diff == 0 or getNode.diff == getNode.depth:
19            stop = time.time()
20            getSolution = True
21            printRoutes(getNode) # print rute dari root ke ditemukannya solusi
22            countNodes -= 1 # kurangi jumlah node yang dibangkitkan karena dihitung root
23            return
24
25    # check daftar valid move pada node ini
26    validMove(getNode.emptyPosRow, getNode.emptyPosCol, getNode.move)
27    for move in listValidMoves:
28        if move == "Move Up":
29            newPosRow, newPosCol = moveUp(getNode.emptyPosRow, getNode.emptyPosCol)
30        if move == "Move Down":
31            newPosRow, newPosCol = moveDown(getNode.emptyPosRow, getNode.emptyPosCol)
32        if move == "Move Left":
33            newPosRow, newPosCol = moveLeft(getNode.emptyPosRow, getNode.emptyPosCol)
34        if move == "Move Right":
35            newPosRow, newPosCol = moveRight(getNode.emptyPosRow, getNode.emptyPosCol)
36
37    # membuat child node
38    childNode = makeNode(getNode.matrix, getNode.emptyPosRow, getNode.emptyPosCol, newPosRow, newPosCol, getNode.depth + 1, getNode, move)
39    # tambahkan child node ke dalam PrioQueue
40    liveNode.push(childNode)
```

Gambar 28 Fungsi Pencarian Solusi



```
1  # fungsi membangkitkan puzzle random
2  def createMatrixRandom():
3      listElement = list(range(1, 17))
4      random.shuffle(listElement)
5      matrix = [[0 for j in range(4)] for i in range(4)]
6      for i in range(4):
7          for j in range(4):
8              matrix[i][j] = listElement[i*4 + j]
9      return matrix
```

Gambar 29 Fungsi Pembangkitan *Random Puzzle*

```

1  # Main
2  # program akan menhandle jika terjadi error pada masukan file
3  try:
4      print("\n15 Puzzle Solver with Branch and Bound Algorithm\n")
5      print("\nDaftar Pilihan Mode :\n1. Random\n2. Input File")
6      mode = int(input("\nPilih mode : "))
7      if mode == 1:
8          print("\nPerhatikan bahwa mode ini dapat membuat looping program sangat banyak\nhingga tidak dapat dilanjutkan!")
9          print("\nApakah anda yakin ingin melanjutkan? (y/n)\n")
10         if input("(y/n) : ") == "y":
11             l = createMatrixRandom()
12         else:
13             print("\nProgram dihentikan\n")
14             exit()
15     if mode == 2:
16         filename = input("\nMasukkan nama file puzzle yang ingin dicari solusinya: ")
17         file = filename + ".txt"
18         # proses pembentukan matriks dari file
19         with open(file, 'r') as f:
20             l = [[int(num) for num in line.split(' ')] for line in f]
21         print("Puzzle yang akan dicari adalah seperti berikut:\n")
22         printMatrix(l)
23         print("Nilai Kurang(i) untuk setiap ubin adalah sebagai berikut:\n")
24         isSolveable(l)
25         if not solveable:
26             print("\nPersoalan ini tidak dapat menghasilkan solusi\n")
27         else:
28             print("\nPersoalan ini dapat diselesaikan dengan langkah sebagai berikut:\n")
29             start = time.time()
30             findSolution(l)
31             if getSolution:
32                 print("Finished!\n\nSimpul dibangkitkan\t=\t", countNodes)
33                 print("Time execution\t\t=\t", stop - start, "sekon\n")
34             else: # program akan berhenti jika tidak waktu sudah mencapai limit
35                 print("\nPersoalan terlalu lama dieksekusi, tidak dapat dilanjutkan\n\nExiting... \n")
36     except:
37         # keluaran jika terjadi error dalam input nama file
38         print("\nFile masukan tidak tersedia, silahkan run program & coba lagi")
39         print("\nExiting... \n")

```

Gambar 30 *Main Program*

IV. Teks Persoalan Data Uji

File *matrix1.txt*

1	2	3	4
5	6	16	8
9	10	7	11
13	14	15	12

File *matrix2.txt*

1	3	4	15
2	16	5	12
7	6	11	14
8	9	10	13

File *matrix3.txt*

```
6 5 4 3
2 1 16 8
9 10 7 11
13 14 15 12
```

File *matrix4.txt*

```
1 2 3 4
5 6 7 8
9 12 15 14
13 11 16 10
```

File *matrix5.txt*

```
3 1 2 4
16 5 7 8
10 6 11 12
9 13 14 15
```

V. Alamat GitHub

<https://github.com/irsyadazka/15PuzzleSolver.git>

VI. Checklist Fitur Program

Poin	Ya	Tidak
1. Program berhasil dikompilasi	√	
2. Program berhasil <i>running</i>	√	
3. Program dapat menerima input dan menuliskan output	√	

4. Luaran sudah benar untuk semua data uji	√	
5. Bonus dibuat		√