LAPORAN TUGAS KECIL

Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch and Bound*

Ditujukan untuk memenuhi salah satu tugas kecil mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2021/2022

Disusun oleh:

Sarah Azka Arief (K2) **13520083**

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

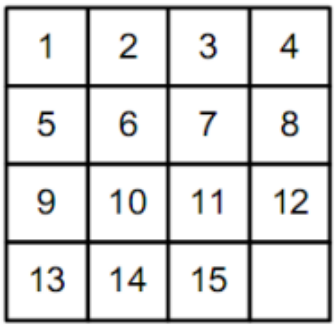
BANDUNG

2022

DAFTAR ISI

1. CARA KERJA PROGRAM

Pada Tugas Kecil 3 IF2211 Strategi Algoritma, digunakan algoritma *branch and bound* untuk menyelesaikan persoalan 15-Puzzle dengan nilai *bound* tiap simpul berupa jumlah *cost* yang dibutuhkan untuk mencapai simpul tersebut dari akar serta taksiran *cost* berupa jumlah ubin tidak kosong yang tidak berada pada tempat sesuai susunan akhir alias *goal state*. *Goal state* sendiri dapat dilihat seperti pada gambar di bawah.

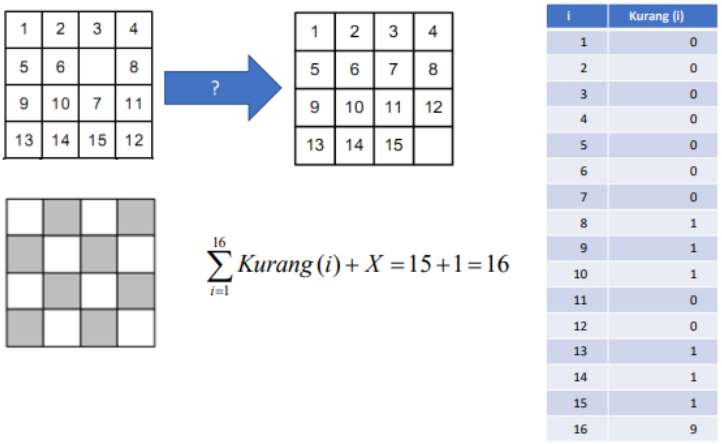


Gambar 1. Goal State 15-Puzzle
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Kecil-3-(2022).pdf

Pada program yang dibuat untuk memenuhi tugas kecil ini, algoritma *branch and bound* diterapkan dengan menggunakan bahasa *python*. Secara umum, program ini dibagi menjadi dua *file* berupa *main program* yang terdapat pada ‘main.py’ serta modul penyelesaian yang menerapkan *branch and bound* yang berada pada *file* ‘solver.py’. *File* ‘solve.py’ terdiri atas beberapa fungsi:

1. matrixInput
   Menerima input *puzzle* secara manual dan mengembalikan 2D array, angka penanda ubin kosong, serta *lower bound* berupa jumlah ubin yang tidak di *goal state*
2. matrixFile
   Membaca *puzzle* pada *file* yang terletak di *folder* ‘test’ dan mengembalikan 2D array, angka penanda ubin kosong, serta *lower bound* berupa jumlah ubin yang tidak di *goal state*
3. getIdx
   Mengembalikan posisi (koordinat x dan y) dari suatu angka yang valid pada ubin *puzzle*
4. reachable
   Menggunakan fungsi KURANG(i) untuk mencari apakah *goal state reachable* dari *initial state*. Mengembalikan *bool* yang menandakan *reachability* dari *goal state*
5. swap
   Menukar ubin kosong dengan ubin yang berada di atas, bawah, kanan, atau kirinya
6. displayMatrix
   Menampilkan *puzzle*
7. search
   Mencari *goal state* dengan menerapkan *branch and bound*

Program ini diterapkan dengan terlebih dahulu menerima *input* matriks berupa *puzzle* yang akan dipecahkan. Ubin pada *puzzle* diisi dengan angka 1 hingga 15, sementara ubin kosong ditandai dengan angka lain selain angka 1 hingga 15. Kemudian, akan dilakukan pengecekan terhadap *reachability* dari *goal state* dengan menggunakan fungsi ‘reachable’ yang memanfaatkan KURANG(i) di bawah.



i	Kurang (i)
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

Gambar 2. Contoh Fungsi KURANG(i)
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Kecil-3-(2022).pdf

Program akan melakukan pengecekan terhadap posisi dari ubin kosong. Apabila ubin kosong berada pada koordinat x ganjil dan y genap atau x genap dan y ganjil maka X adalah 1, sedangkan untuk sebaliknya X adalah 0. Kemudian dilakukan iterasi untuk setiap ubin untuk mnghitung kurang(i) dengan cara mencari jumlah angka yang lebih kecil dari angka pada ubin i yang posisinya ubinnya melebihi ubin i. Apabila didapat total dari seluruh kurang(i) adalah angka genap, maka *goal state reachable*. Di lain hal, apabila didapat angka ganjil, maka *goal state* tidak *reachable*.

Apabila *goal state reachable*, dilakukan pencarian dengan menggunakan fungsi 'search'. Fungsi 'search' menyimpan *dictionary* bernama 'dir' yang berisi arah koordinat x, arah koordinat y, serta string berupa nama arah dari setiap pergerakan ubin yang valid. Fungsi tersebut juga menginisialisasi *built-in priority queue* dari *python* yang diisi dengan *initial state* dari *puzzle* serta *node* yang dibangkitkan selama proses *search. Priority queue* ini akan diisi dengan *array* yang berisi informasi relevan dari setiap *node* yakni *cost*, *state* dari *puzzle*, jumlah ubin yang tidak berada pada posisi, kedalaman, serta *path* yang ditempuh untuk mencapai *node* tersebut. Perlu dicatat bahwa *path* disimpan dalam suatu *array* yang berisi *state* of *puzzle* serta string yang berupa arah pergerakan untuk mencapai *state* tersebut.

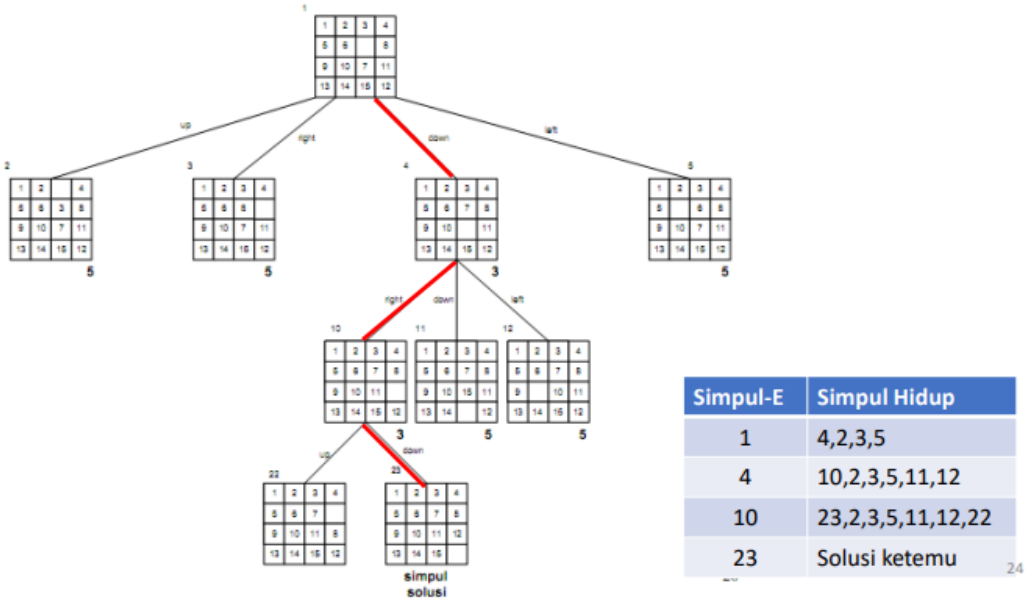
Pertama-tama, fungsi 'search' akan menginisialisasi *priority queue* dengan *initial state.* Setelahnya, fungsi akan memasuki *loop* yang baru akan berhenti apabila *goal state* telah dicapai. Di dalam *loop*, pertama-tama akan diambil *node* pada *priority queue* yang memiliki *cost* terkecil. Adapun *cost* dari suatu *node* dihitung dengan persamaan berikut, dengan *f(P)* brupa panjang lintasan dari simpul akar ke P serta *g(P)* berupa taksiran panjang lintasan terpendek dari P ke simpul solusi dengan akar P:

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

Gambar 3. Taksiran cost simpul P pada 15-Puzzle
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Kecil-3-(2022).pdf

Pada program ini, optimasi dilakukan dengan *f(P)* didapat dengan menghitung kedalaman dari *node* tersebut alias menambahkan 1 dari kedalaman *parent node*nya. Selain itu, *g(P)* diambil dengan memanfaatkan *lower bound*. Dalam setiap pergerakan ubin kosong, taksiran panjang lintasan terpendk dapat bertambah 1 apabila ubin yang akan ditukar dengan ubin kosong sudah berada pada *goal state*-nya, berkurang 1 apabila *goal state* dari ubin yang akan ditukar dengan ubin kosong terletak pada posisi ubin kosong, atau tidak bertambah maupun berkurang apabila ubin yang akan ditukar tetap tidak pada *goal state*-nya saat maupun sebelum ditukar dengan ubin kosong.

Setelah didapat *node* dengan *cost* terkecil, dilakukan pencarian terhadap pergerakan ubin kosong yang valid. Untuk setiap arah yang dicoba, apabila pergerakan tersebut valid maka *cost* dari *child node* tersebut akan diinisialisasi dengan *cost* dari *node* serta *path* dari *child node* akan berupa *path* dari *node* yang ditambahkan *child node* pada akhir *path*. Hal ini berlangsung hingga dicapai *goal state*. Saat *goal state* telah tercapai, akan ditampilkan *elapsed time*, *path* yang ditempuh untuk mencapai *goal state*, serta jumlah *node* yang dikunjungi, dibangkitkan, dan dibutuhkan untuk mencapai *goal state*.



Simpul-E	Simpul Hidup
1	4,2,3,5
4	10,2,3,5,11,12
10	23,2,3,5,11,12,22
23	Solusi ketemu

Gambar 4. Visualisasi output
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Kecil-3-(2022).pdf

2. SOURCE CODE PROGRAM

2.1 main.py

```python
from solver import matrixInput, matrixFile, reachable, getIdx, search

print("\n 15 PUZZLE SOLVER")
print("                \n")

print("-------------------------------------------------------------\n")
print("PILIHAN INPUT PUZZLE")
print("1. Input Manual")
print("2. Input File")
choice = int(input("Pilih input puzzle: "))
# invalid input
while(choice != 1 and choice != 2):
    print("Pilihan tidak tersedia. Coba lagi")
print("\n-------------------------------------------------------------\n")
print("INPUT PUZZLE")
# input user
if (choice == 1):
    puzzle, blank, lowerbound = matrixInput()
# input file
else:
    string = input("Enter nama file: ")
    puzzle, blank, lowerbound = matrixFile(string)
print("\n-------------------------------------------------------------\n")
# check reachability
x, y = getIdx(puzzle, blank)
reachability = reachable(puzzle, blank, x, y)
print("\n-------------------------------------------------------------\n")
# search
if (reachability):
    search(puzzle, blank, lowerbound)
else:
    print("GOAL STATE UNREACHABLE!")

```

2.2 solver.py

```python
import copy
import time
import numpy as np
from queue import PriorityQueue

'''
get starting puzzle matrix from input
'''
def matrixInput():
    blank = -1
    checkPos = 1
    lowerbound = 0
    puzzle = [[0 for i in range(4)] for j in range(4)]
    # fill puzzle matrix
    for i in range(4):
        for j in range(4):
            num = int(input("Enter element (" + str(i) + "," + str(j) + "): "))
            # check for blank tile
            if (num < 1 or num > 15):
                blank = num
            else:
                # check if already in solved position
                if (num != checkPos):
                    lowerbound += 1
            checkPos += 1
            puzzle[i][j] = num
    return puzzle, blank, lowerbound
```

```python
'''
get starting puzzle matrix from file
'''
def matrixFile(filename):
    puzzle = []
    blank = -1
    checkPos = 1
    lowerbound = 0
    filename = "test/" + filename
    with open(filename) as f:
        lines = f.readlines()
        for line in lines:
            temp = []
            # fill puzzle matrix
            for num in line.split():
                num = int(num)
                # check for blank tile
                if (num < 1 or num > 15):
                    blank = num
                else:
                    # check if already in solved position
                    if (num != checkPos):
                        lowerbound += 1
                temp.append(num)
                checkPos += 1
            puzzle.append(temp)
        f.close()
    return puzzle, blank, lowerbound


'''
get element index
'''
def getIdx(puzzle, val):
    temp = np.array(puzzle)
    res = np.where(temp == val)
    return (res[0][0]), (res[1][0]) # returns x and y position


'''
check reachability using kurang(i) function
'''
def reachable(puzzle, blank, x, y):
    # check if blank tile is in shaded area
    if (((x % 2 == 1) and (y % 2 == 0)) or ((x % 2 == 0) and (y % 2 == 1))):
        kurangx = 1
    else:
        kurangx = 0
    # initialize 1D array of puzzle elements in order
    check = sum(puzzle, [])
    check.remove(blank)
    total = 0
    kurangi = {}
    for row in range(4):
        for col in range(4):
            i = puzzle[row][col]
            if (i == blank):
                i = 16 # convert blank tile value to 16
            else:
                check.remove(i) # remove value from checked
            # get kurang(i)
            kurangsum = sum(j < i for j in check)
            kurangi[i] = kurangsum
            total += kurangsum
    for i in range(16):
        print("KURANG(%02d): " % (i+1) + str(kurangi[i+1]))
    res = kurangx + total
    print("SUM KURANG(i) + X: " + str(res))
    # check if reachable
    if (res % 2 == 0):
        return True
    else:
        return False
```

```python
'''
move blank tile in a certain direction by swapping with a target tile
'''
def swap(currnode, dirX, dirY, blank, visited):
    # initialize solved position
    inPos = {
        1: [0, 0], 2: [0, 1], 3: [0, 2], 4: [0, 3], 5: [1, 0],
        6: [1, 1], 7: [1, 2], 8: [1, 3], 9: [2, 0], 10: [2, 1],
        11: [2, 2], 12: [2, 3], 13: [3, 0], 14: [3, 1], 15: [3, 2]
    }
    node = copy.deepcopy(currnode)
    x, y = getIdx(node, blank)
    swapX = x + dirX
    swapY = y + dirY
    # check if movement is invalid (out of range)
    if (swapX > 3 or swapX < 0 or swapY > 3 or swapY < 0):
        return -1, -1
    pos = inPos[(node[swapX][swapY])]
    # case when target tile is already in solved position
    if (pos[0] == swapX and pos[1] == swapY):
        addCost = 1
    # case when target tile is not in solved position and will be in solved position
    elif (pos[0] == x and pos[1] == y):
        addCost = -1
    # case when target tile is and will not be in solved position
    else:
        addCost = 0

    # swap blank tile with target tile
    temp = node[x][y]
    node[x][y] = node[swapX][swapY]
    node[swapX][swapY] = temp

    # ignore node if redundant (already visited)
    if node in visited:
        return -1, -1
    return node, addCost


'''
display puzzle matrix
'''
def displayMatrix(nodes, blank):
    count = 0
    for node in nodes:
        # print puzzle state description
        if (count == 0):
            print("INITIAL STATE")
        elif (count == 1):
            print("MOVES TAKEN")
        elif (count == len(nodes) - 1):
            print("FINAL STATE")
        # print movement direction
        print ("DIRECTION: " + node[1])
        # print puzzle
        print("┌────┬────┬────┬────┐")
        array = node[0] # initialize array
        for i in range(4):
            for j in range(4):
                print("|",end="")
                if (array[i][j] == blank):
                    print("  ", end=" ") # print blank tile as blank (no number)
                else:
                    print("%02d" % (array[i][j]), end=" ")
            print("|")
            if(i != 3):
                print("├────┼────┼────┼────┤")
        print("└────┴────┴────┴────┘\n")
        count += 1
```

```python
'''
branch and bound search
'''
def search(puzzle, blank, lowerbound):
    # initialize movement direction
    dir = { 0: [-1, 0, "UP"], 1: [1, 0, "DOWN"], 2: [0, -1, "LEFT"], 3: [0, 1, "RIGHT"] }

    visited = []
    generated = 0
    currpath = [[puzzle, "-"]]
    livenode = PriorityQueue()
    livenode.put([lowerbound, puzzle, lowerbound, 0, currpath]) # add initial puzzle state

    start = time.time()
    while (True):
        # get lowest cost node
        toSearch = livenode.get()
        # initialize node info
        currcost = toSearch[0]
        currnode = toSearch[1]
        currleft = toSearch[2]
        currdepth = toSearch[3]
        currpath = toSearch[4]
        visited.append(currnode)

        # case when all tiles are in solved position
        if (currleft == 0):
            end = time.time()
            displayMatrix(currpath, blank)
            print("Elapsed time: " + str(end - start))
            print("Total step: " + str(currdepth))
            print("Visited nodes: " + str(len(visited) - 1))
            print("Generated nodes: " + str(generated))
            break

        # get nodes for all movement directions
        for i in range(4):
            pos = dir[i]
            nodeDir, addCost = swap(currnode, pos[0], pos[1], blank, visited)
            # case when node is valid (not out of range or visited)
            if (nodeDir != -1):
                costDir = currcost + 1 + addCost # initialize cost
                path = currpath + [[nodeDir, (dir[i])[2]]] # initialize path and movement
                livenode.put([costDir, nodeDir, currleft + addCost, currdepth + 1, path])
                generated += 1
```

3. HASIL PERCOBAAN

3.1 Tidak Dapat Diselesaikan

Nama File	unsolvable1.txt
Initial State Puzzle	15 2 1 12 8 5 6 11 4 9 10 7 3 14 13 16

Hasil

```
PS C:\Users\Sarah Azka A\Documents\if\sem-4\tubes\fifteen-puzzle-solver> python main.py
15 PUZZLE SOLVER
---------------------------------------------------------

PILIHAN INPUT PUZZLE
1. Input Manual
2. Input File
Pilih input puzzle: 2

---------------------------------------------------------

INPUT PUZZLE
Enter nama file: unsolvable1.txt

---------------------------------------------------------

KURANG(01): 0
KURANG(02): 1
KURANG(03): 0
KURANG(04): 1
KURANG(05): 2
KURANG(06): 2
KURANG(07): 1
KURANG(08): 5
KURANG(09): 2
KURANG(10): 2
KURANG(11): 5
KURANG(12): 9
KURANG(13): 0
KURANG(14): 1
KURANG(15): 14
KURANG(16): 0
SUM KURANG(i) + X: 45

---------------------------------------------------------

GOAL STATE UNREACHABLE!
PS C:\Users\Sarah Azka A\Documents\if\sem-4\tubes\fifteen-puzzle-solver>
```

3.2 Tidak Dapat Diselesaikan

Nama File	Unsolvable2.txt
Initial State Puzzle	1 3 4 15 2 16 5 12 7 6 11 14 8 9 10 13

Hasil

```
PS C:\Users\Sarah Azka A\Documents\if\sem-4\tubes\fifteen-puzzle-solver> python main.py
15 PUZZLE SOLVER
---------------------------------------------------------

PILIHAN INPUT PUZZLE
1. Input Manual
2. Input File
Pilih input puzzle: 2

---------------------------------------------------------

INPUT PUZZLE
Enter nama file: unsolvable2.txt

---------------------------------------------------------

KURANG(01): 0
KURANG(02): 0
KURANG(03): 1
KURANG(04): 1
KURANG(05): 0
KURANG(06): 0
KURANG(07): 1
KURANG(08): 0
KURANG(09): 0
KURANG(10): 0
KURANG(11): 3
KURANG(12): 6
KURANG(13): 0
KURANG(14): 4
KURANG(15): 11
KURANG(16): 10
SUM KURANG(i) + X: 37

---------------------------------------------------------

GOAL STATE UNREACHABLE!
PS C:\Users\Sarah Azka A\Documents\if\sem-4\tubes\fifteen-puzzle-solver>
```

3.3 Dapat Diselesaikan

Nama File	solvable1.txt
Initial State Puzzle	1 2 3 4 6 10 7 8 5 13 15 11 9 0 14 12

Hasil

```
PS C:\Users\Sarah Azka A\Documents\if\sem-4\tubes\fifteen-puzzle-solver>
```

```
15 PUZZLE SOLVER

-----------------------------------------------------

PILIHAN INPUT PUZZLE
1. Input Manual
2. Input File
Pilih input puzzle: 2

-----------------------------------------------------

INPUT PUZZLE
Enter nama file: solvable1.txt

-----------------------------------------------------

KURANG(01): 0
KURANG(02): 0
KURANG(03): 0
KURANG(04): 0
KURANG(05): 0
KURANG(06): 1
KURANG(07): 1
KURANG(08): 1
KURANG(09): 0
KURANG(10): 4
KURANG(11): 1
KURANG(12): 0
KURANG(13): 3
KURANG(14): 1
KURANG(15): 4
KURANG(16): 2
SUM KURANG(i) + X: 18

-----------------------------------------------------
```

INITIAL STATE
DIRECTION: -

01	02	03	04
06	10	07	08
05	13	15	11
09		14	12

MOVES TAKEN
DIRECTION: UP

01	02	03	04
06	10	07	08
05		15	11
09	13	14	12

DIRECTION: UP

01	02	03	04
06		07	08
05	10	15	11
09	13	14	12

DIRECTION: LEFT

01	02	03	04
	06	07	08
05	10	15	11
09	13	14	12

DIRECTION: DOWN

01	02	03	04
05	06	07	08
	10	15	11
09	13	14	12

DIRECTION: DOWN

01	02	03	04
05	06	07	08
09	10	15	11
	13	14	12

DIRECTION: RIGHT

01	02	03	04
05	06	07	08
09	10	15	11
13		14	12

DIRECTION: RIGHT

01	02	03	04
05	06	07	08
09	10	15	11
13	14		12

DIRECTION: UP

01	02	03	04
05	06	07	08
09	10		11
13	14	15	12

DIRECTION: RIGHT

01	02	03	04
05	06	07	08
09	10	11	
13	14	15	12

FINAL STATE
DIRECTION: DOWN

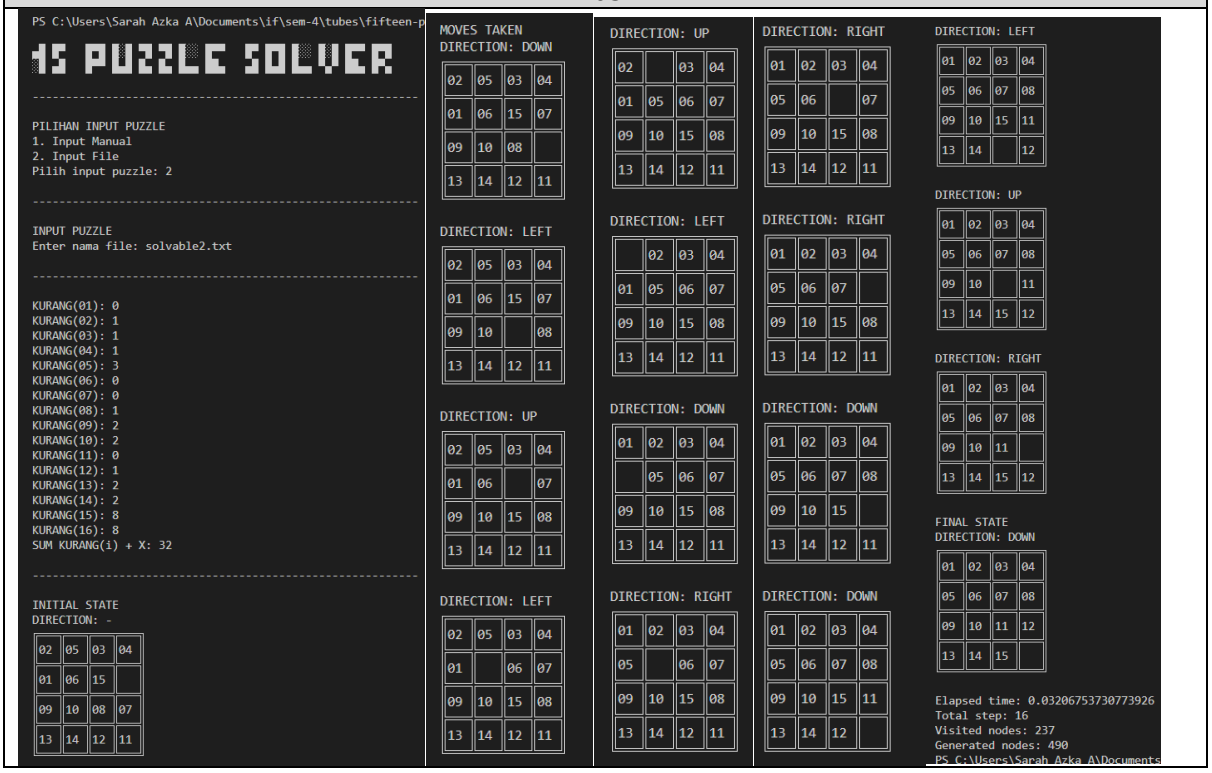
01	02	03	04
05	06	07	08
09	10	11	12
13	14	15	

```
Elapsed time: 0.008054018020629883
Total step: 10
Visited nodes: 19
Generated nodes: 42
PS C:\Users\Sarah Azka A\Documents\if\sem-4\tubes\fifteen
```

3.4 Dapat Diselesaikan

Nama File	solvable2.txt
Initial State Puzzle	2 5 3 4 1 6 15 0 9 10 8 7 13 14 12 11

Hasil



3.5 Dapat Diselesaikan

Nama File	solvable3.txt
Initial State Puzzle	2 3 4 11 1 5 10 8 9 6 12 15 13 14 16 7

Hasil

LAMPIRAN

Link Github: https://github.com/azkazkazka/Tucil3_13520083

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		✓

Solvable1.txt

1 2 3 4

6 10 7 8

5 13 15 11

9 0 14 12

Solvable2.txt

2 5 3 4

1 6 15 0

9 10 8 7

13 14 12 11

Solvable3.txt

2 3 4 11

1 5 10 8

9 6 12 15

13 14 16 7

Unsolvable1.txt

15 2 1 12

8 5 6 11

4 9 10 7

3 14 13 16

Unsolvable2.txt

1 3 4 15

2 16 5 12

7 6 11 14

8 9 10 13