LAPORAN TUGAS KECIL
IF2211 – STRATEGI ALGORITMA

ALGORITMA PENYELESAIAN PERSOALAN 15-PUZZLE DENGAN METODE BRANCH AND BOUND

Dipersiapkan oleh:

Dzaky Fattan Rizqullah

13520003

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESHA 10, BANDUNG 40132

Daftar Isi

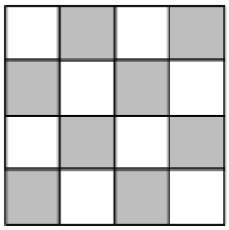
A. Algoritma *Branch and Bound*

Program ini dibuat untuk dapat menyelesaikan persoalan 15-*puzzle* yang diberikan melalui file .txt yang diletakkan pada folder test. Algoritma yang digunakan adalah *Branch and Bound*. Secara garis besar, program akan meminta input file .txt yang sudah terletak pada folder test, kemudian akan menampilkan kondisi awal matriks persoalan, lalu menampilkan nilai dari fungsi Kurang(i) yang dapat menentukan apakah persoalan dapat diselesaikan atau tidak. Bila dapat diselesaikan, program akan berusaha menyelesaikan *puzzle* dengan algoritma yang sudah disebutkan di atas. Bila persoalan telah diselesaikan, program akan menampilkan langkah-langkah penyelesaian *puzzle* dari kondisi awal hingga kondisi terselesaikannya *puzzle*.

Fungsi Kurang(i) adalah fungsi yang menentukan apakah suatu *state* awal matriks dapat diselesaikan. Teoremanya, status tujuan hanya dapat dicapai dari status awal jika

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

Bernilai genap. Nilai X sama dengan satu jika sel kosong pada posisi awal berada pada sel yang diarsir pada matriks di bawah.



Gambar 1. Matriks puzzle yang diarsir

Fungsi Kurang(i) sama dengan banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i). POSISI(i) = posisi ubin bernomor i pada susunan yang diperiksa. Fungsi ini diimplementasikan dalam program sebagai `KURANGFunc(matrix)`. Nilai fungsi Kurang untuk masing-masing i ditampilkan beserta nilai dari $\sum_{i=1}^{16} KURANG(i) + X$.

Nilai *bound* tiap simpul adalah pernjumlahan antara dua nilai, yaitu sebagai berikut:

1. C*ost* yang diperlukan untuk sampai suatu simpul x dari akar, yang diimplementasikan dalam program sebagai `f_func(matrix)`.
2. Taksiran *cost* jumlah ubin tidak kosong yang tidak berada pada tempat sesuai susunan akhir (*goal state*). Fungsi ini diimplementasikan dalam program sebagai `g_func(matrix)`.

Setelah dipastikan bahwa matriks awal dapat diselesaikan, selanjutnya dilakukan penyelesaian *puzzle* menggunakan algoritma *Branch and Bound*. *State* matriks merupakan simpul dari *Tree* untuk *Branch and Bound*, dengan *state* awal matriks merupakan simpul akar. Fungsi ini diimplementasikan dalam program sebagai fungsi `solve()`. Ada beberapa objek yang diinisialisasi, yaitu sebagai berikut.

1. `start_state`, bernilai matriks *state* awal,
2. `goal_state`, bernilai matriks *state* akhir,
3. *Timer*, mengukur waktu eksekusi program,

4. Sebuah *Priority Queue* untuk menyimpan simpul matriks yang akan dibangkitkan anak-anaknya dinisialisasi dengan elemen awal yaitu *state* awal matriks,
5. Sebuah *list* `visited` yang berisi *simpul* matriks yang sudah pernah dicapai, implementasinya memanfaatkan library `heapq` dari Python.
6. Sebuah *list* `matrix_step` yang berisi *list* tahap penyelesaian matriks,
7. Sebuah *dictonary* dengan *key* berupa matriks yang diubah menjadi *string* dan *value*-nya adalah *state* matriks sebelum dikenai suatu gerakan yang menghasilkan *state* matriks *key* (menghubungkan simpul *parent* dengan simpul *child*),
8. Sebuah *dictonary* dengan *key* berupa matriks yang diubah menjadi *string* dan *value*-nya adalah sebuah *tuple* yang berisi gerakan menuju matriks tersebut dan nilai `f_func` yang bersesuaian,

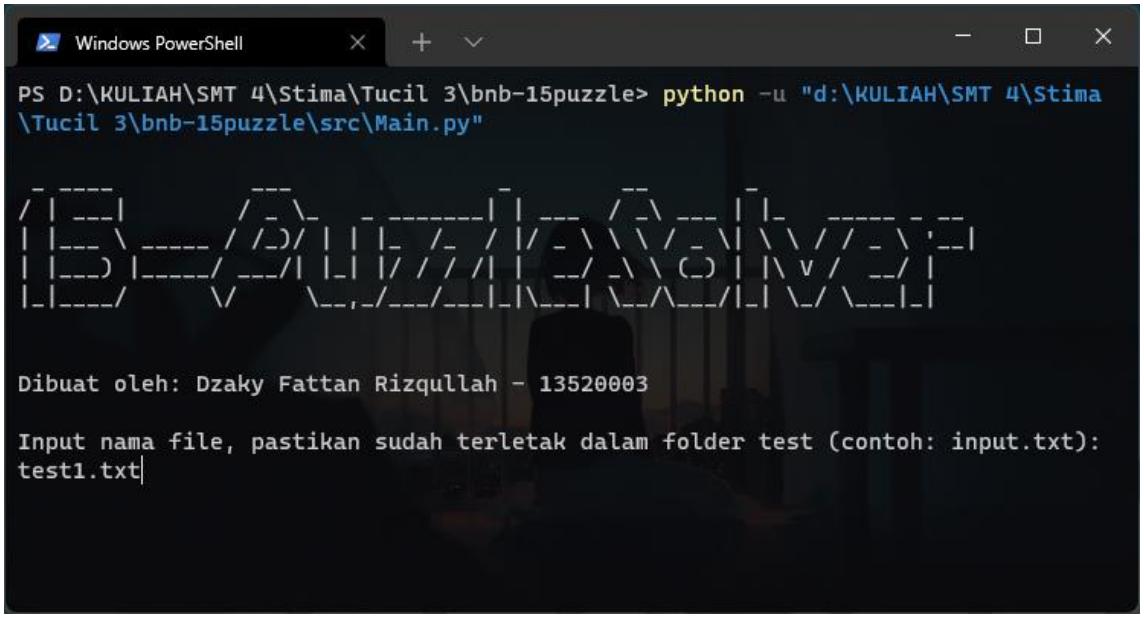
Satu per satu matriks pada *Priority Queue* di-*pop* dan dimasukkan ke dalam list `visited`, yang mana keseluruhannya dibangkitkan keempat kemungkinan perubahan yang dapat dilakukan, yaitu menggerakan sel kosong ke atas, bawah, kiri, dan kanan, dengan mempertimbangkan validitas gerakan (tidak dapat menggeser sel kosong ke bawah bila sel kosong terletah di baris paling bawah matriks). Keempat matriks ini dicari nilai *cost*-nya yang merupakan penjumlahan nilai `f_func dan g_func`. Selanjutnya keempat matriks ini di-*push* ke *Priority Queue* dengan memprioritaskan nilai *cost* matriks yang lebih kecil. Keempat matriks ini dibuat *dictionary*-nya dengan properti seperti yang dijelaskan sebelumnya. Setelah itu semua, matriks selanjutnya pada *Priority Queue* di-*pop* dan dibangkitkan lagi anak-anaknya seperti matriks sebelumnya.

Proses di atas di-*loop* hingga jawaban didapatkan atau waktu eksekusi melebihi batas. Kalang (*loop*) didesain sedemikian rupa agar program tidak akan memproses *state* matriks yang sudah didatangi (dibangkitkan anak-anaknya). Proses ini dapat dilakukan dengan melakukan pengecekan keberadaan *state* matriks pada list `visited`. Waktu eksekusi sengaja dibuat karena fungsi heuristik memiliki kemungkinan dapat membuat program terjebak dalam *infinite loop*, dikarenakan fungsi heuristik yang digunakan sebenarnya tidak cukup baik untuk kasus dengan jumlah gerakan yang cukup banyak (30 gerakan atau lebih). Waktu eksekusi dibatasi selama 1200 detik.

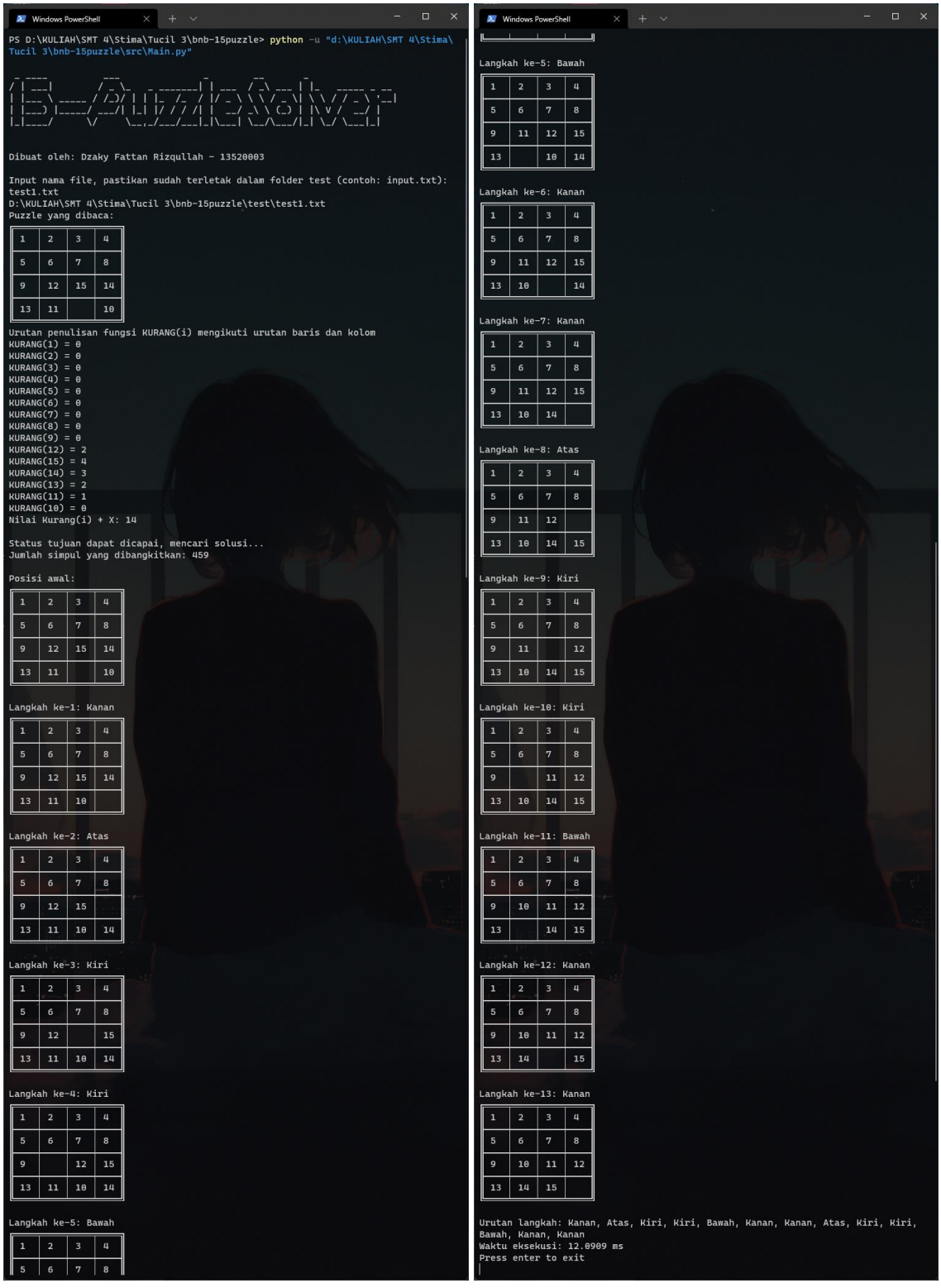
Bila jawaban didapatkan, maka *list* `matrixStep` mulai diisi dengan meng-*push* matriks *goal state*, selanjutnya merunut *parent*-nya dengan menggunakan *dictionary* yang sudah dijelaskan semuanya, hingga ke matriks awal. Selanjutnya program menampilkan keluaran berupa Elemen-elemen pada `matrixStep` ditampilkan sebagai urutan langkah penyelesaian. Urutan langkah penyelesaian (gerakan atas, bawah, kiri, atau kanan) serta waktu eksekusi juga ditampilkan setelahnya.

B. *Screenshot Input* dan *Output* Program

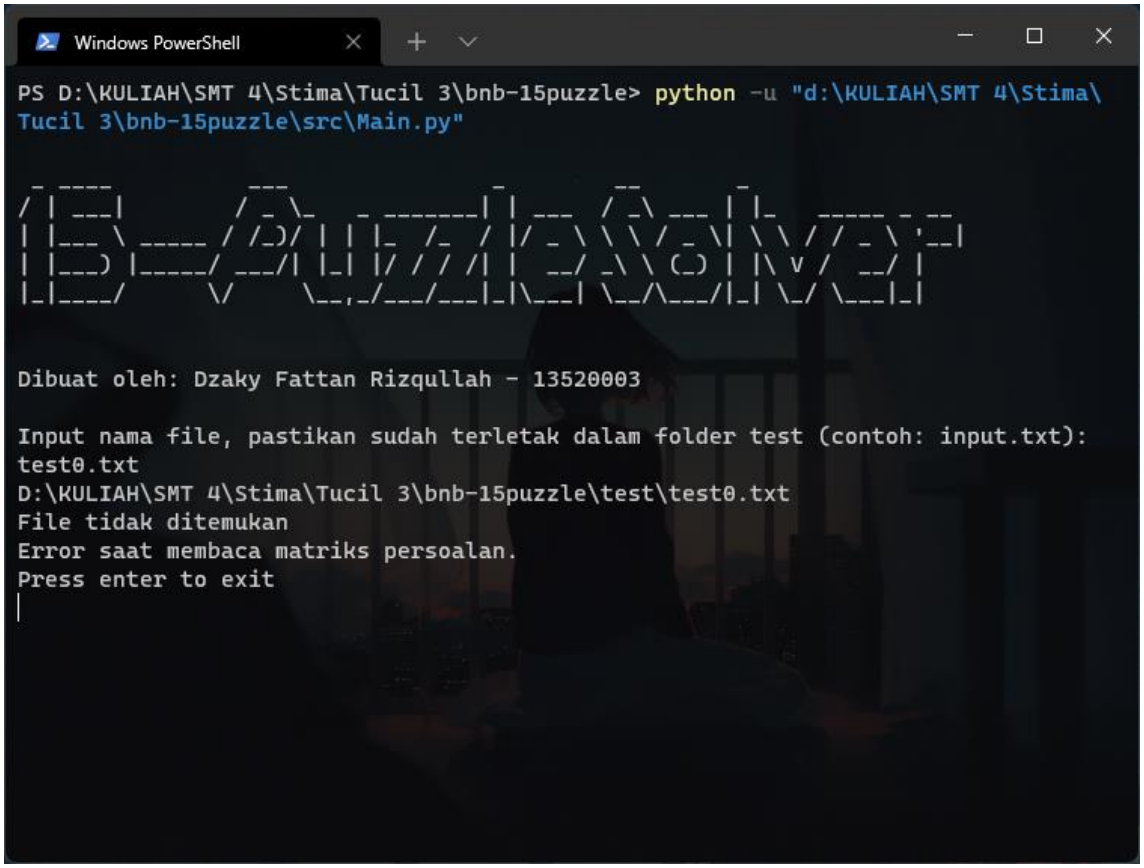
Berikut dilampirkan tangkapan layar *input* serta *output* program. Untuk contoh kasusnya dijelaskan lebih lanjut pada subbab E. Perlu diperhatikan bahwa *bonus* (pembuatan *GUI* yang menampilkan animasi urutan penyelesaian *puzzle*) tidak diimplementasikan untuk tugas kecil ini.



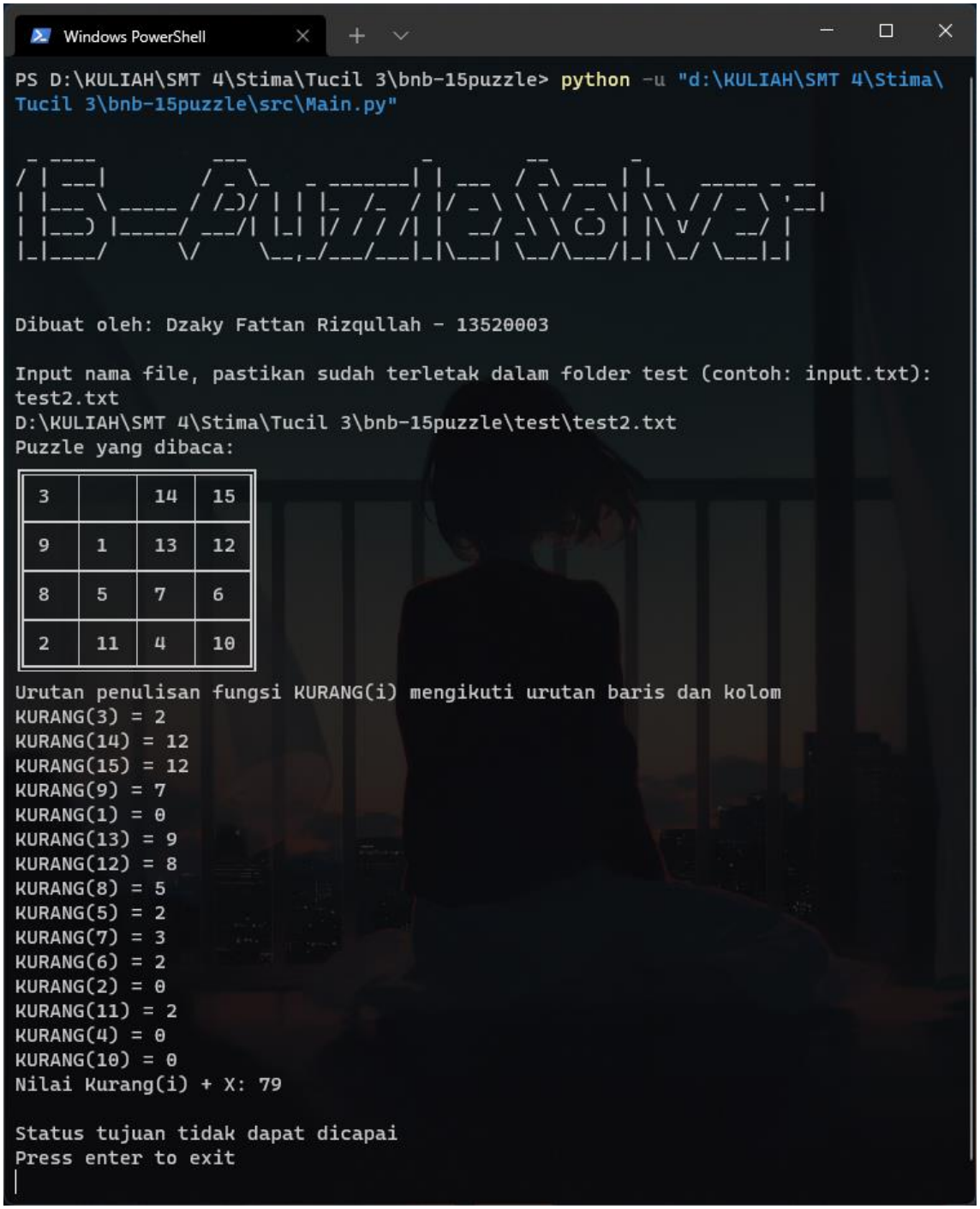
Gambar 2. Antarmuka awal, dengan masukan file test1.txt



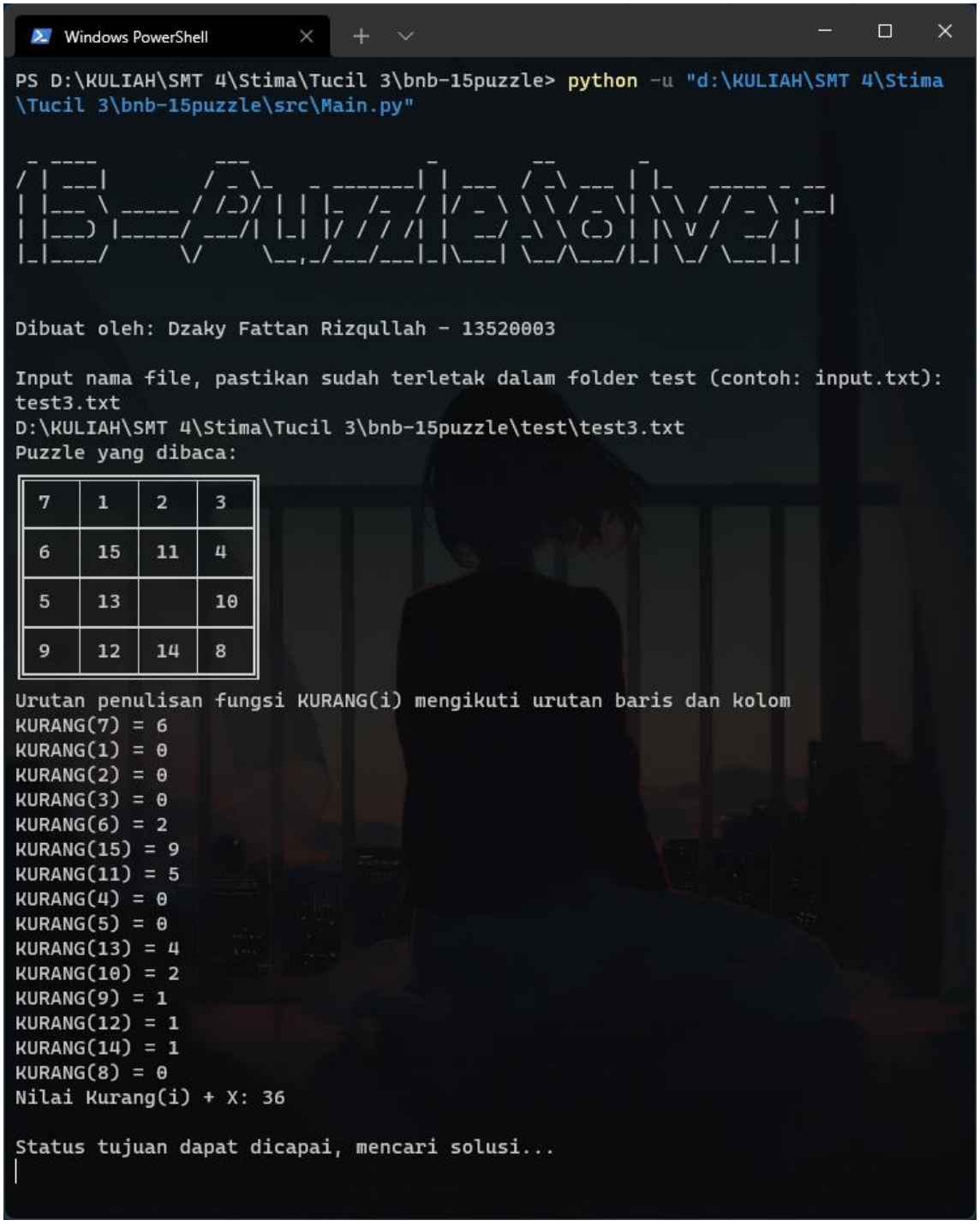
Gambar 3. Proses dan hasil pencarian solusi untuk file test1.txt



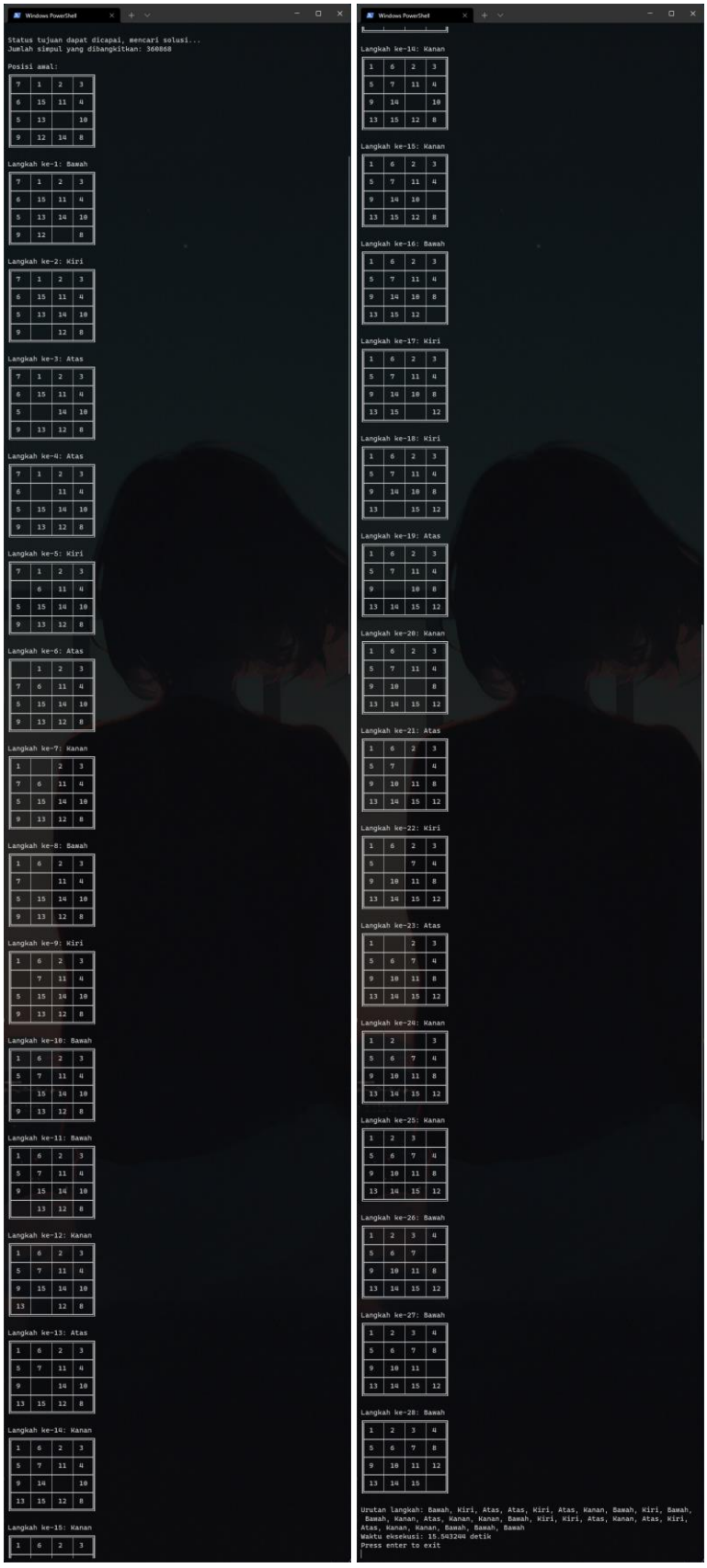
Gambar 4. Antarmuka awal, dengan masukan file test0.txt yang berujung tidak dapat ditemukan



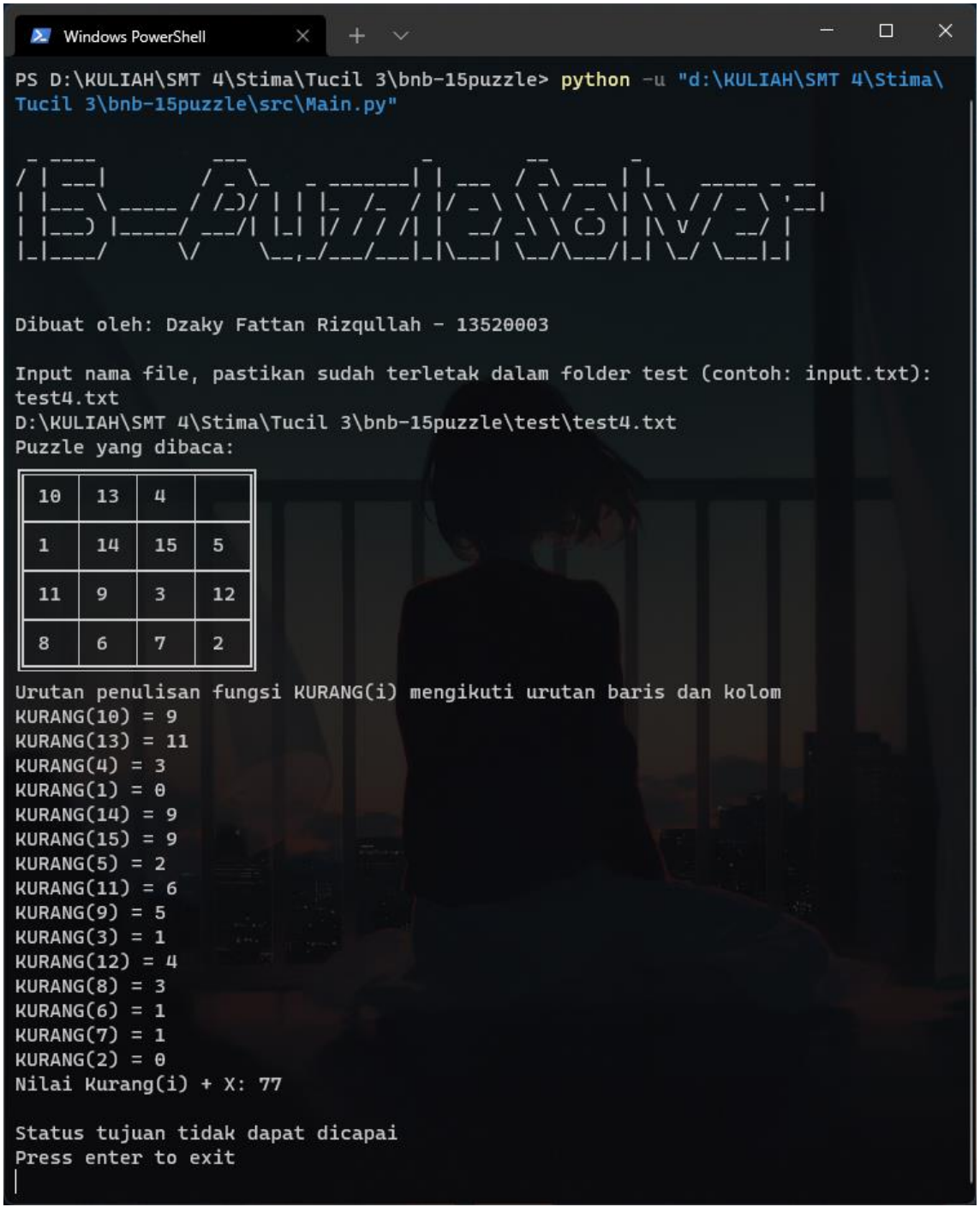
Gambar 5. Proses pencarian solusi untuk file test2.txt yang berujung gagal karena status tujuan tidak dapat dicapai



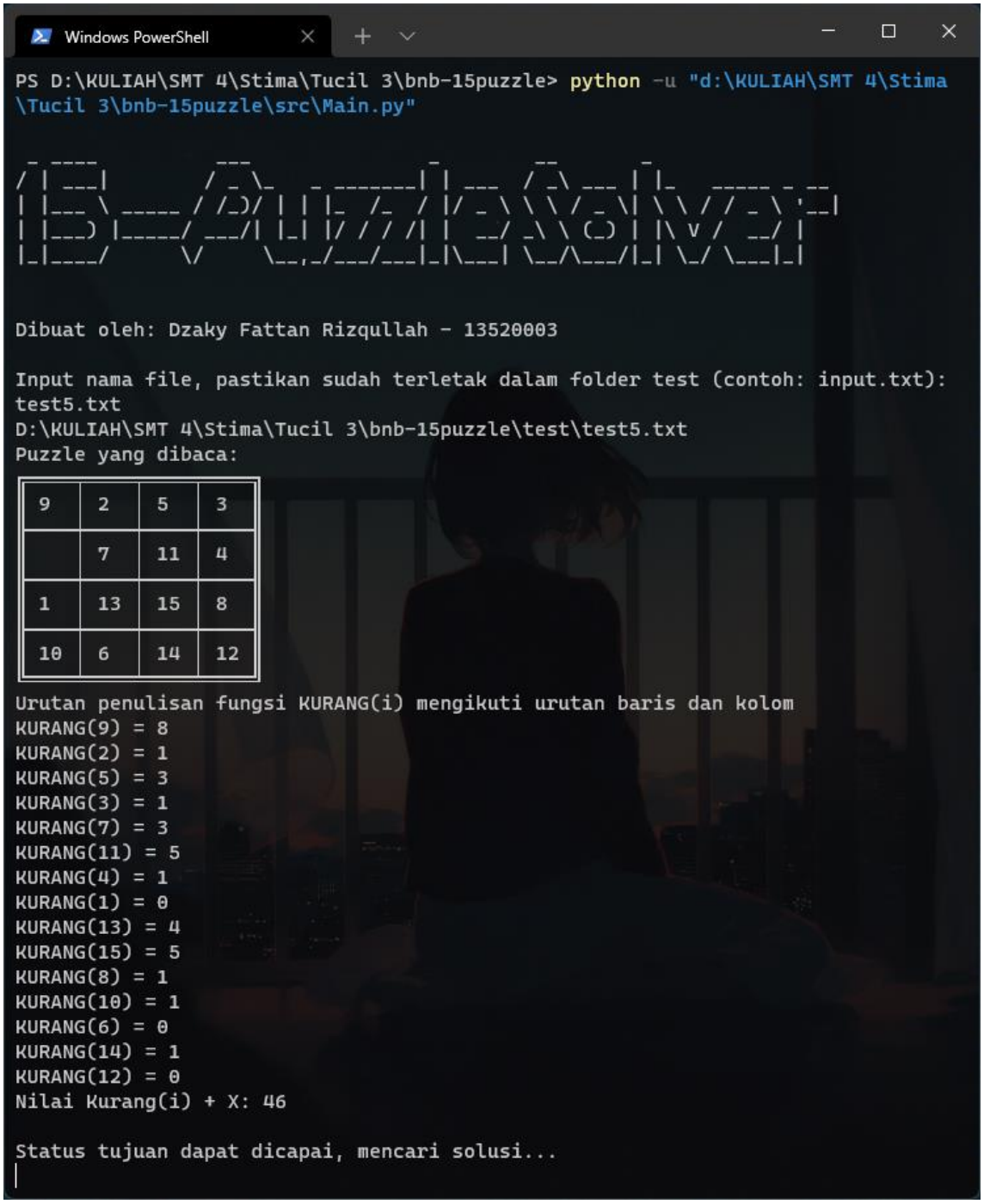
Gambar 6. Proses pencarian solusi untuk file test3.txt



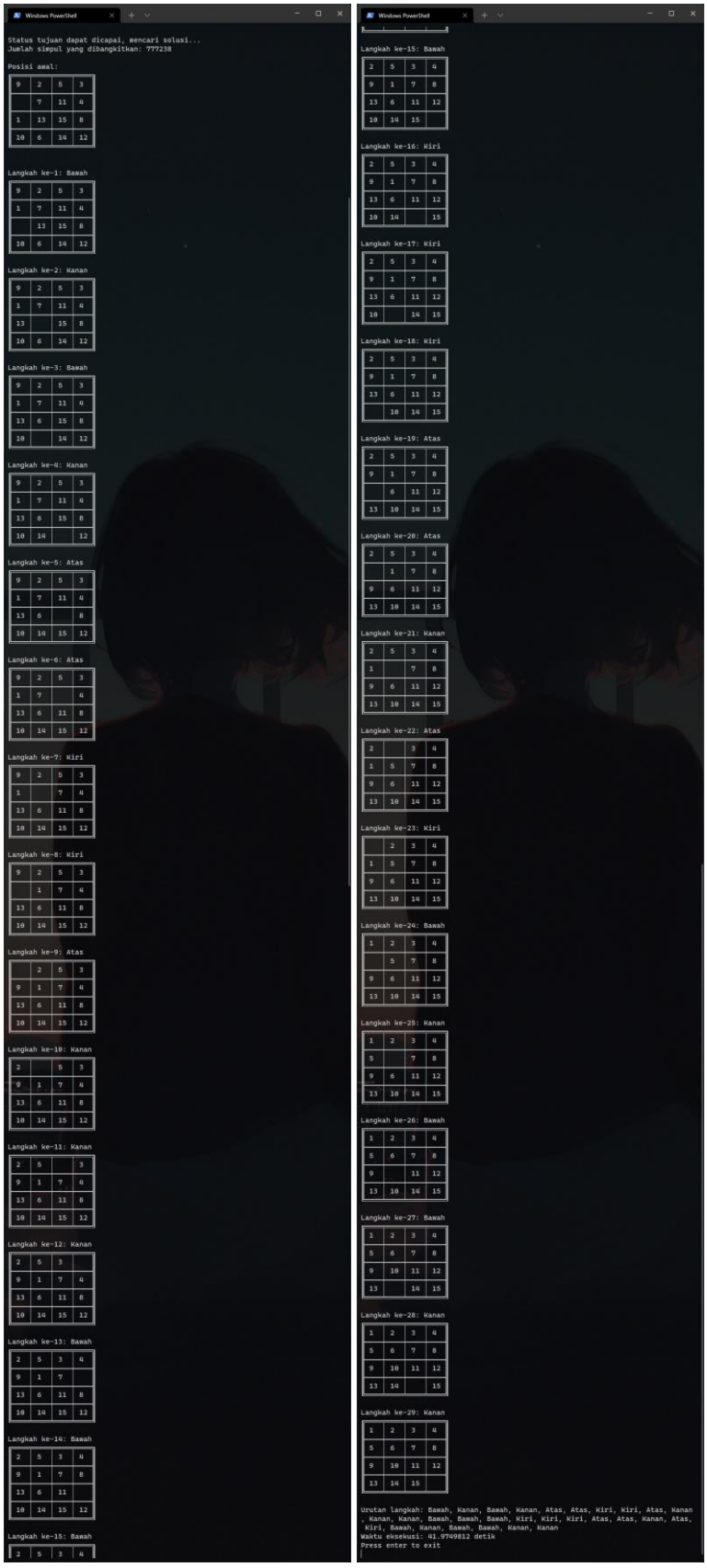
Gambar 7. Hasil pencarian solusi untuk file test3.txt



Gambar 8. Proses pencarian solusi untuk file test4.txt yang berujung gagal karena Status tujuan tidak dapat dicapai



Gambar 9. Proses pencarian solusi untuk file test5.txt



Gambar 10. Hasil pencarian solusi untuk file test5.txt

C. Check List

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

Berikut dilampirkan kode program yang ditulis dalam bahasa Python 3. Program dibagi menjadi tiga file, yang akan dijabarkan masing-masing di bawah.

`MatrixController.py`, merupakan file yang berisi fungsi manipulasi matriks yang merupakan representasi 15-puzzle.

```
import os

# read matrix from .txt file from specified input
def read_matrix(filename):
    matrix = []
    # get parent folder path
    cwd = os.getcwd()
    if (os.path.basename(os.path.normpath(cwd)) != "bnb-15puzzle"):
        pardir = os.path.abspath(os.path.join(cwd, os.pardir))
    else:
        pardir = cwd
    fileDir = os.path.join(pardir, "test", filename)
    print(fileDir)
    try:
        with open(fileDir, 'r') as f:
            for line in f:
                tempList = list(line.strip().split())
                tempLine = []
                for x in tempList:
                    try:
                        tempLine.append(int(x))
                    except ValueError:
                        tempLine.append(16)
                matrix.append(tempLine)
                # matrix.append([int(x) for x in tempList])
        return matrix
    except FileNotFoundError:
        print("File tidak ditemukan")
        return None
```

```
# print matrix properly
def print_matrix(matrix):
    if (matrix != None):
        print("╔═══╤═══╤═══╤═══╗")
        # ┌ ┌   ┌   ┌─  ─   │   │
        for i in range(len(matrix)):
            print("╟", end=" ")
            for j in range(len(matrix[i])):
                strToPrint = str(matrix[i][j]) + " " if matrix[i][j] < 10 else
str(matrix[i][j])
                if strToPrint == "16": strToPrint = "  "
                strToPrint = strToPrint + " │" if j != len(matrix[i])-1 else
strToPrint + " ╟"
                print(strToPrint, end=" ")
            if i != len(matrix) - 1:
                print("\n╟───┼───┼───┼───╢")
        print("\n╚═══╧═══╧═══╧═══╝")
    else:
        print("Matrix kosong")

# find 16 position in matrix, return -1, -1 if not found
def find_16(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if (matrix[i][j] == 16):
                return i, j
    return -1, -1

# copy matrix
def copy_matrix(matrix):
    tempMatrix = []
    for i in range(len(matrix)):
        tempMatrix.append([])
        for j in range(len(matrix[i])):
            tempMatrix[i].append(matrix[i][j])
    return tempMatrix

# move 16 to specified position
def move_16(matrix, direction):
    i, j = find_16(matrix)
    tempMatrix = copy_matrix(matrix)
    if direction == "Atas":
        if i > 0:
            tempMatrix[i][j], tempMatrix[i-1][j] = matrix[i-1][j], matrix[i][j]
    elif direction == "Bawah":
        if i < len(matrix)-1:
```

```
            tempMatrix[i][j], tempMatrix[i+1][j] = matrix[i+1][j], matrix[i][j]
    elif direction == "Kiri":
        if j > 0:
            tempMatrix[i][j], tempMatrix[i][j-1] = matrix[i][j-1], matrix[i][j]
    elif direction == "Kanan":
        if j < len(matrix[i])-1:
            tempMatrix[i][j], tempMatrix[i][j+1] = matrix[i][j+1], matrix[i][j]
    return tempMatrix

# transform matrix to string
def mat_to_str(matrix):
    strToReturn = ""
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            strToReturn += str(matrix[i][j]) + " "
    return strToReturn.strip()
```

`Solver.py`, merupakan file yang berisi fungsi-fungsi tempat diimplementasikannya algoritma *Branch and Bound*.

```
from time import time
import MatrixController as m
import heapq as hq

# goal state
goal_state = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]

start_state = []
timeLimit = 1200

# dictionary to store the parent of each matrixState
parent = {}
# dictionary to store matrix move and lvl
mat_info = {}

# enumerate direction
direction = ["Atas", "Bawah", "Kiri", "Kanan"]

# Function Kurang(i)
def kurang_func(matrix):
    kurang = 0
    emptyPos = -1
    print("Urutan penulisan fungsi KURANG(i) mengikuti urutan baris dan kolom")
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
```

```
            if matrix[i][j] == 16:
                emptyPos = i + j
            tempKurang = 0
            for k in range(i * len(matrix) + j + 1, len(matrix)**2):
                if matrix[k//len(matrix)][k % len(matrix)] < matrix[i][j]:
                    tempKurang += 1
            if (matrix[i][j] != 16):
                print("KURANG(" + str(matrix[i][j]) + ") = " + str(tempKurang))
            kurang += tempKurang
    return kurang if emptyPos % 2 == 0 else kurang+1

# fFunc to find length from root parent to current matrix
def f_func(matrix):
    return mat_info[m.mat_to_str(matrix)][1]

# our heuristic function, the number of misplaced tiles
def g_func(matrix):
    g = 0
    for i in range(len(matrix)**2):
        if matrix[i//len(matrix)][i % len(matrix)] !=
goal_state[i//len(matrix)][i % len(matrix)]:
            g += 1
    return g

# create matrix_step from the goal state to the start state
def connect_parent(newMatrix, matrix_step):
    global parent
    str_mat = m.mat_to_str(newMatrix)
    matrix_step.append((newMatrix, mat_info[str_mat][0]))
    tempPar = parent[str_mat]
    while tempPar != start_state:
        if (tempPar, mat_info[m.mat_to_str(newMatrix)][0]) not in matrix_step:
            matrix_step.append((tempPar, mat_info[m.mat_to_str(tempPar)][0]))
            tempPar = parent[m.mat_to_str(tempPar)]
    matrix_step.append(start_state)
    matrix_step.reverse()

# solver
def solve(matrix, matrix_step):
    # matrix stores the current matrixState
    # matrix_step stores the steps of solving the matrix

    # set start_state
    global start_state
    start_state = matrix
```

```
    # parent stores the parent of each matrixState
    global parent

    # visited stores the visited matrixState
    visited = set()

    # f(x)
    mat_info[m.mat_to_str(matrix)] = ("Init", 0)
    level = 0

    # live_queue stores the list of matrixState that are not yet visited,
    # is a PrioQueue to make sure we visited the matrixState with the lowest gFunc
    live_queue = []
    hq.heapify(live_queue)

    # a timer, to prevent a very long execution time
    start = time()

    # if start state is already a goal state, return the matrix_step
    if matrix == goal_state:
        matrix_step.append(matrix)
        return

    # push first matrix state to live_queue
    hq.heappush(live_queue, (f_func(matrix) + g_func(matrix), matrix))
    count = 0
    while live_queue and time() - start < timeLimit:
        matrix = live_queue.pop(0)
        matrix = matrix[1]
        # check if matrix already visited
        matstr = m.mat_to_str(matrix)
        if matstr in visited:
            continue
        visited.add(matstr)

        if matrix != start_state:
            level = mat_info[m.mat_to_str(matrix)][1]
        level += 1

        # iterate all 4 possible direction (build node)
        for move in enumerate(direction):
            newMatrix = m.move_16(matrix, move[1])
            matstr = m.mat_to_str(newMatrix)
            # skips if it's already visited, prevent from going backwards
            if (matstr in visited):
                continue
```

```
            # print("Jumlah simpul dibangkitkan: " + str(count))
            # create dict of matrix info and the parent
            mat_info[matstr] = (move[1],  level)
            parent[matstr] = matrix

            # count++
            count += 1

            # check if newMatrix is the goal state
            if newMatrix == goal_state:
                connect_parent(newMatrix, matrix_step)
                print("Jumlah simpul yang dibangkitkan: " + str(count))
                return

            # else, queue the newMatrix
            hq.heappush(live_queue, (level + g_func(newMatrix), newMatrix))

    # if time limit exceded
    if live_queue:
        connect_parent(matrix, matrix_step)
        print("Jumlah simpul yang dibangkitkan: " + str(count))
```

`Main.py`, merupakan program utama yang menjalankan fungsi interaksi dengan pengguna. Masukan dan keluaran program di-*handle* oleh file ini.

```
from time import time_ns
import MatrixController as m
import Solver as s

# list of steps
matrix_step = []

def print_title():
    print("""
 _ ____          ___                _         __       _
/ | ___|        / _ \_   _ _______| | ___   / _\ ___ | |_   _____ _ __
| |___ \ _____ / /_)/ | | |_  /_  / |/ _ \ \ \ / _ \| \ \ / / _ \ '__|
| |___) |_____/ ___/| |_| |/ / / /| |  __/ _\ \ (_) | |\ V /  __/ |
|_|____/      \/     \__,_/___/___|_|\___| \__/\___/|_| \_/ \___|_|

""")
    print("\nDibuat oleh: Dzaky Fattan Rizqullah - 13520003\n")

def calc_time(start, end):
```

```
    duration = (end - start)
    if duration > 1000000000:
        duration = duration / 1000000000
        print("\nWaktu eksekusi:", duration, "detik")
    elif duration > 1000000:
        duration = duration / 1000000
        print("\nWaktu eksekusi:", duration, "ms")
    else:
        print("\nWaktu eksekusi:", duration, "ns")

# The Main Program
def main():
    print_title()
    ipt = input("\nInput nama file, pastikan sudah terletak dalam folder test
(contoh: input.txt): ")
    matrix = m.read_matrix(ipt)
    if (matrix != None):
        print("Puzzle yang dibaca:")
        m.print_matrix(matrix)
        kurang = s.kurang_func(matrix)
        print("Nilai Kurang(i) + X:", kurang)
        if kurang % 2 == 1:
            print("\nStatus tujuan tidak dapat dicapai")
        else:
            start = time_ns()
            print("\nStatus tujuan dapat dicapai, mencari solusi...")
            s.startState = matrix
            s.solve(matrix, matrix_step)
            end = time_ns()
            if (matrix_step != []):
                print("\nPosisi awal:")
                m.print_matrix(matrix_step.pop(0))
                for i in range(len(matrix_step)):
                    print("\nLangkah ke-" + str(i+1) + ": " +
str(matrix_step[i][1]))
                    m.print_matrix(matrix_step[i][0])
            print("\nUrutan langkah: ", end="")
            for i in range(len(matrix_step)):
                str_to_print = str(matrix_step[i][1]) + ", " if i !=
len(matrix_step) - 1 else str(matrix_step[i][1])
                print(str_to_print, end="")
            calc_time(start, end)
    else:
        print("Error saat membaca matriks persoalan.")
    # prompt before exiting
    input("Press enter to exit\n")
```

```
if __name__ == "__main__":
    main()
```

E. Berkas Persoalan

Berikut dilampirkan lima buat persoalan 15-puzzle yang digunakan sebagai contoh kasus untuk tugas kecil ini. Tiga di antaranya (`test1.txt`, `test3.txt`, `test5.txt`) dapat diselesaikan, sementara dua lainnya (`test2.txt` dan `test4.txt`) tidak dapat diselesaikan. Keseluruhan file persoalan diletakkan dalam folder test

File test	**test1.txt**	**test2.txt**	**test3.txt**	**test4.txt**	**test5.txt**
Matriks	1 2 3 4 5 6 7 8 9 12 15 14 13 11 16 10	3 16 14 15 9 1 13 12 8 5 7 6 2 11 4 10	7 1 2 3 6 15 11 4 5 13 - 10 9 12 14 8	10 13 4 16 1 14 15 5 11 9 3 12 8 6 7 2	9 2 5 3 - 7 11 4 1 13 15 8 10 6 14 12
Waktu eksekusi	±12.1 ms	-	±15.5 ms	-	±41.9 s
Jumlah gerak	13	-	28	-	29
Simpul dibangkitkan	459	-	360868	-	777238

F. Alamat Drive dan Github

Drive:	https://drive.google.com/drive/folders/1DQ0uRJXf9d5Kjcs_h8G9EMOLFURw5wt4?usp=sharing (Link di atas akan menampilkan drive berisi kode program, executable, testcase, serta file laporan ini. Pastikan diakses dengan akun fakultas.)
Github:	https://github.com/DzakyFattan/bnb-15puzzle