Laporan Tugas Kecil 3

IF2211 Strategi Algoritma

Semester II Tahun 2021/2022

disusun oleh:

Muhammad Fikri Ranjabi 13520002

K-02

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

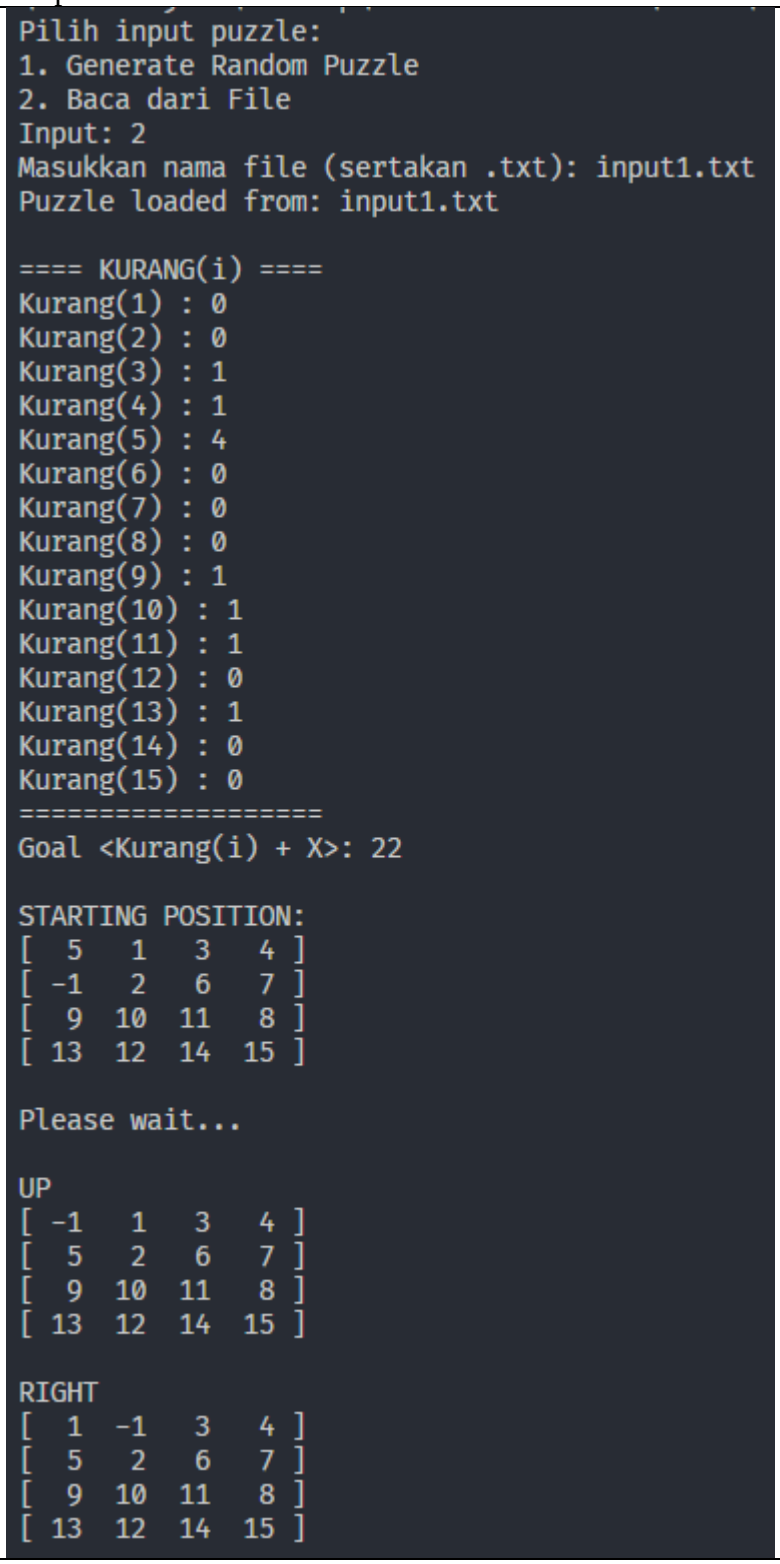
BANDUNG 2022

a. Alur Kerja Program

1. Membaca input dari file teks atau dari puzzle yang diacak.
2. Instansiasi setiap elemen dari objek puzzle dari bacaan puzzle beserta posisi elemen kosong pada puzzle.
3. Mulai execution time.
4. Tambahkan posisi awal (akar dari pohon) sebagai elemen pertama pada queue, jika posisi tersebut adalah solusi (goal) maka solusi telah ditemukan. Pop elemen queue.
5. Jika queue kosong maka stop pencarian.
6. Jika queue tidak kosong, bangkitkan simpul langkah-langkah berikutnya dalam fungsi pembatas.
7. Hitung cost dari simpul yang dibangkitkan dan tambahkan simpul baru tersebut ke queue.
8. Kembali ke langkah 5.
9. Jika persoalan dapat diselesaikan dan solusi telah ditemukan, maka tampilkan execution time, posisi awal 15-puzzle, nilai dari fungsi kurang, nilai goal, jumlah simpul yang dibangkitkan, dan urutan matriks dari posisi awal ke posisi akhir.
10. Jika persoalan tidak dapat diselesaikan berdasarkan hasil dari fungsi kurang, maka tampilkan nilai dari fungsi kurang, nilai goal, dan pesan “15-puzzle tidak dapat diselesaikan”.

b. Screenshot Input Output Program

Berikut adalah hasil dari test case dari file input1.txt, input2.txt, input3.txt yang dapat diselesaikan dan input4.txt, input5.txt merupakan persoalan yang tidak bisa diselesaikan.

No	Output
1	

```
Pilih input puzzle:
1. Generate Random Puzzle
2. Baca dari File
Input: 2
Masukkan nama file (sertakan .txt): input1.txt
Puzzle loaded from: input1.txt

==== KURANG(i) ====
Kurang(1) : 0
Kurang(2) : 0
Kurang(3) : 1
Kurang(4) : 1
Kurang(5) : 4
Kurang(6) : 0
Kurang(7) : 0
Kurang(8) : 0
Kurang(9) : 1
Kurang(10) : 1
Kurang(11) : 1
Kurang(12) : 0
Kurang(13) : 1
Kurang(14) : 0
Kurang(15) : 0
===================
Goal <Kurang(i) + X>: 22

STARTING POSITION:
[  5   1   3   4 ]
[ -1   2   6   7 ]
[  9  10  11   8 ]
[ 13  12  14  15 ]

Please wait...

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Puzzle telah selesai.
Execution Time: 0.0800318717956543 s
Node Generated: 661
```

2

```
Pilih input puzzle:
1. Generate Random Puzzle
2. Baca dari File
Input: 2
Masukkan nama file (sertakan .txt): input2.txt
Puzzle loaded from: input2.txt

==== KURANG(i) ====
Kurang(1) : 0
Kurang(2) : 0
Kurang(3) : 1
Kurang(4) : 1
Kurang(5) : 0
Kurang(6) : 2
Kurang(7) : 4
Kurang(8) : 1
Kurang(9) : 0
Kurang(10) : 3
Kurang(11) : 1
Kurang(12) : 0
Kurang(13) : 1
Kurang(14) : 4
Kurang(15) : 1
===================
Goal <Kurang(i) + X>: 24

STARTING POSITION:
[  1   3   7   4 ]
[  6   2  10   8 ]
[  5  14  -1  11 ]
[  9  13  15  12 ]

Please wait...

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

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

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

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

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

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

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

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

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

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

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

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

Puzzle telah selesai.
Execution Time: 0.010004043579101562 s
Node Generated: 71
```

3	Pilih input puzzle: 1. Generate Random Puzzle 2. Baca dari File Input: 2 Masukkan nama file (sertakan .txt): input3.txt Puzzle loaded from: input3.txt ==== KURANG(i) ==== Kurang(1) : 0 Kurang(2) : 0 Kurang(3) : 0 Kurang(4) : 1 Kurang(5) : 4 Kurang(6) : 0 Kurang(7) : 2 Kurang(8) : 1 Kurang(9) : 4 Kurang(10) : 0 Kurang(11) : 0 Kurang(12) : 0 Kurang(13) : 1 Kurang(14) : 1 Kurang(15) : 1 ========== Goal <Kurang(i) + X>: 22 STARTING POSITION: [5 1 2 4] [9 7 3 8] [-1 6 10 11] [13 14 15 12] Please wait... UP [5 1 2 4] [-1 7 3 8] [9 6 10 11] [13 14 15 12] UP [-1 1 2 4] [5 7 3 8] [9 6 10 11] [13 14 15 12] RIGHT [1 -1 2 4] [5 7 3 8] [9 6 10 11] [13 14 15 12]	

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

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

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

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

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

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

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

Puzzle telah selesai.
Execution Time: 0.008002758026123047 s
Node Generated: 119
```

4	Pilih input puzzle: 1. Generate Random Puzzle 2. Baca dari File Input: 2 Masukkan nama file (sertakan .txt): input4.txt Puzzle loaded from: input4.txt ==== KURANG(i) ==== Kurang(1) : 0 Kurang(2) : 0 Kurang(3) : 1 Kurang(4) : 1 Kurang(5) : 0 Kurang(6) : 0 Kurang(7) : 3 Kurang(8) : 2 Kurang(9) : 0 Kurang(10) : 7 Kurang(11) : 1 Kurang(12) : 0 Kurang(13) : 1 Kurang(14) : 7 Kurang(15) : 1 ========== Goal <Kurang(i) + X>: 31 STARTING POSITION: [1 3 10 4] [7 2 14 8] [5 -1 6 11] [9 13 15 12] Please wait... Puzzle tidak dapat diselesaikan	

5	Pilih input puzzle: 1. Generate Random Puzzle 2. Baca dari File Input: 2 Masukkan nama file (sertakan .txt): input5.txt Puzzle loaded from: input5.txt ==== KURANG(i) ==== Kurang(1) : 0 Kurang(2) : 0 Kurang(3) : 0 Kurang(4) : 2 Kurang(5) : 2 Kurang(6) : 0 Kurang(7) : 4 Kurang(8) : 2 Kurang(9) : 1 Kurang(10) : 9 Kurang(11) : 1 Kurang(12) : 3 Kurang(13) : 5 Kurang(14) : 4 Kurang(15) : 3 ================== Goal <Kurang(i) + X>: 49 STARTING POSITION: [10 1 -1 4] [7 5 2 8] [13 3 14 12] [15 9 11 6] Please wait... Puzzle tidak dapat diselesaikan	

c. Status Pengerjaan

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

File read.py

```
def run(filename):
    f = open(filename, "r")
    list = [[0 for i in range(4)] for j in range(4)]
    line = 0
    col = 0
    for x in f:
        col = 0
        for y in x.split():
            list[line][col] = y
            col+= 1
            if (col == 4):
                break
        line += 1

    return list

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

File main.py

```
from copy import deepcopy
import read
import time
import random

# Global variable
nodeNumber = 1
```

```
depth = 0
path = []

class Elmt:
    def __init__(self, pos):
        self.val = -1
        self.pos = pos

class Puzzle:
    def __init__(self, emptyPos):
        self.name = 1
        self.elmt = [[Elmt( (i+1) + (4*j) ) for i in range(4)] for j
in range(4)]
        self.fp = 0
        self.gp = 0
        self.cost = 0
        self.emptyPos = emptyPos
        self.emptyPosRow = (emptyPos-1)//4
        self.emptyPosCol = (emptyPos-1)%4
        self.nextMove = {"up": True, "right": True, "down": True,
"left": True}
        self.prevMove = ""
        self.path = []

    def elmtToList(self):
        list = [[-1 for i in range(4)] for j in range(4)]
        for i in range(4):
            for j in range(4):
                list[i][j] = self.elmt[i][j].val
        return list


    def incrNode(self):
        global nodeNumber
        nodeNumber += 1

    def rowColToPos(self, row, col):
    # convert 4x4 index to 0 index
        return ((row+1) + (4*col))

    def posToRowCol(self, n):
    # convert 1 index to 4x4 index
        row = (n-1)//4
        col = (n-1)%4
        return row, col

    def findEmptyMatrix(self, flatten):
    # return empty position
```

```
        for i in range(4):
            for j in range(4):
                if self.elmt[i][j].val == -1:
                    if flatten:
                        # 1 index
                        return self.elmt[i][j].pos
                    else:
                        # 4x4 index
                        return self.posToRowCol(self.elmt[i][j].pos)

    def findPos(self, val):
    # return 1 index of val
        pos = -99
        for i in range(4):
            for j in range(4):
                if self.elmt[i][j].val == val:
                    pos = self.elmt[i][j].pos
        if val == 16:
            return 16
        else:
            return pos

    def getKurang(self, iparam):
    # return the number of inversion of iparam
        count = 0
        for i in range(4):
            for j in range(4):
                if (self.elmt[i][j].val < iparam) and
(self.elmt[i][j].pos > self.findPos(iparam) and self.elmt[i][j].val
!= -1):
                    count += 1
        return count

    def getKurang16(self):
    #  return the number of inversion of 16 pos
        count = 0
        for i in range(4):
            for j in range(4):
                if (self.elmt[i][j].val < 16) and
(self.elmt[i][j].pos > self.findPos(-1) and self.elmt[i][j].val != -
1):
                    count += 1
        return count

    def isArsirPos(self):
    # return true if empty position on arsir area
        isArsir = False
```

```
        if ((self.emptyPos == 2) or (self.emptyPos == 4) or
(self.emptyPos == 5) or (self.emptyPos == 7) or (self.emptyPos == 10)
or (self.emptyPos == 12) or (self.emptyPos == 13) or (self.emptyPos
== 15)):
            return True
        return isArsir

    def getGoal(self):
    # return goal
        print("\n==== KURANG(i) ====")
        x = 0
        sumKurang = 0
        if self.isArsirPos()==True:
            x = 1
        for i in range(1, 16):
            print("Kurang(" + str(i) + ") : " +
str(self.getKurang(i)))
            sumKurang += self.getKurang(i)
        lastKurang = self.getKurang16()
        self.goal = (sumKurang+x+lastKurang)
        print("====================")
        print("Goal <Kurang(i) + X>: " + str(self.goal))
        return (sumKurang+x+lastKurang)

    def initCost(self):
    # init new cost
        self.cost = self.fp + self.gp

    def printNextMove(self):
    # print next move
        for key in self.nextMove:
            print(key, ":", self.nextMove[key])

    def getVal(self, i, j):
    # get val at index i, j
        return self.elmt[i][j].val

    def getPos(self, i, j):
    # get pos at index i, j
        return self.elmt[i][j].pos

    def posToRow(self, n):
    # convert n on 1 index to row on 0 index
        return ((n-1)//4)

    def posToCol(self, n):
    # convert n on 1 index to col on 0 index
        return ((n-1)%4)
```

```
    def print(self):
    # print current node puzzle
        # print("---------" + str(self.name) + "---------")
        for i in range(4):
            for j in range(4):
                if j == 0:
                    print("[", end="")

                output = self.elmt[i][j].val
                if len(str(self.elmt[i][j].val)) == 1:
                    output = " " + str(self.elmt[i][j].val)
                print("", output, "", end="")
                if j == 3:
                    print("]", end="")
                if ((j+1) % 4 == 0):
                    print("")
        # print("fp:", self.fp)
        # print("gp:", self.gp)
        # print("cost:", self.cost)
        # print("empty pos:", self.emptyPos)
        # print("empty pos row:", self.emptyPosRow)
        # print("empty pos col:", self.emptyPosCol)
        # print("prev move:", self.prevMove)
        # self.printNextMove()
        # print("====================\n")

    def getWrongPos(self):
    # return the number of wrong position
        countWrongPos = 0
        for i in range(4):
            for j in range(4):
                # print(i, j)
                if self.getVal(i,j) != self.getPos(i,j) and
self.getVal(i,j) != -1:
                    countWrongPos += 1
        return countWrongPos

    def isFinish(self):
    # return true if puzzle is finished
        finish = True
        for i in range(4):
            for j in range(4):
                if self.elmt[i][j].val != self.elmt[i][j].pos and
self.getVal(i,j) != -1:
                    return False
        return finish
```

```
    def findEmpty(self, flatten):
    # return index for empty position
        for i in range(4):
            for j in range(4):
                if self.elmt[i][j].val == -1:
                    if flatten:
                        # 1 index
                        return self.elmt[i][j].pos
                    else:
                        # 4x4 index
                        return self.posToRowCol(self.elmt[i][j].pos)

    def move(self, direction):
    # move empty pos to certain direction
        row = self.emptyPosRow
        col = self.emptyPosCol
        moveRow = 0
        moveCol = 0
        if direction=="up":
            moveRow = -1
            moveCol = 0
        if direction=="down":
            moveRow = 1
            moveCol = 0
        if direction=="left":
            moveRow = 0
            moveCol = -1
        if direction=="right":
            moveRow = 0
            moveCol = 1

        # exchange empty pos with move position
        self.emptyPos =
self.rowColToPos(self.emptyPosRow+moveRow,self.emptyPosCol+moveCol)
        self.emptyPosRow = self.emptyPosRow+moveRow
        self.emptyPosCol = self.emptyPosCol+moveCol

        newPos = self.rowColToPos(row+moveRow,col+moveCol)
        emptyRow, emptyCol = self.findEmpty(False)

        temp = self.elmt[emptyRow+moveRow][emptyCol+moveCol].val
        self.elmt[emptyRow+moveRow][emptyCol+moveCol].val = -1
        self.elmt[emptyRow][emptyCol].val = temp

    def checkMove(self):
    # check all possible move
        for key in self.nextMove:
            self.nextMove[key] = True
```

```
        row, col = self.findEmpty(False)
        if (col == 3 or col == 4):
            self.nextMove["right"] = False
        if (col == 0):
            self.nextMove["left"] = False
        if (row == 0):
            self.nextMove["up"] = False
        if (row == 3 or row == 4):
            self.nextMove["down"] = False
        if self.prevMove != "":
            if self.prevMove == "up":
                self.nextMove["down"] = False
            if self.prevMove == "down":
                self.nextMove["up"] = False
            if self.prevMove == "left":
                self.nextMove["right"] = False
            if self.prevMove == "right":
                self.nextMove["left"] = False

    def __lt__(self, other):
    # overload less than operator for two matrix
        return (self.cost < other.cost) and (self.name < other.name)

    def generateNextMove(self, queue):
    # generate next move and add it to queue
        global depth
        global nodeNumber
        global path
        depth += 1

        self.checkMove()

        for key in self.nextMove:
            if self.nextMove[key] == True:
                temp = deepcopy(self)
                temp.move(key)
                temp.checkMove()
                temp.prevMove = key
                nodeNumber += 1
                temp.name = nodeNumber
                temp.fp += 1
                temp.gp = temp.getWrongPos()
                temp.initCost()
                temp.path.append([key, temp.elmtToList()])
                queue.append((temp.cost, temp))
                nodeNumber += 1
        queue.sort(reverse=True)
```

```
def printQueue(queue):
    for i in range(len(queue)):
        print("(cost(as queue
key):",queue[i][0],",nodeName:",queue[i][1].name,
queue[i][1].prevMove, "parent",queue[i][1].parent)


def main(puzzle):
    global depth
    global path
    startTime = time.time()
    depth = 0

    print("\nSTARTING POSITION:")
    puzzle.print()
    print("\nPlease wait...\n")

    if (puzzle.goal % 2) != 0:
        print("Puzzle tidak dapat diselesaikan")
    else:
        queue = []
        queue.append((puzzle.cost, puzzle))
        if (puzzle.isFinish()):
            head = queue.pop()
            return puzzle
        while (len(queue) != 0):
            head = queue.pop()
            if head[1].isFinish():
                endTime = time.time()
                path = head[1].path
                for k in range(len(path)):
                    print(str(path[k][0]).upper())
                    for i in range(4):
                        for j in range(4):
                            if j == 0:
                                print("[", end="")

                            output = path[k][1][i][j]
                            if len(str(output)) == 1:
                                output = " " + str(output)
                            print("", output, "", end="")
                            if j == 3:
                                print("]", end="")
                            if ((j+1) % 4 == 0):
                                print("")
                    print()
                print("Puzzle telah selesai.")
                print("Execution Time:", endTime - startTime, "s")
```

```
                print("Node Generated:", nodeNumber)
                print()

                return head[1]
            else:
                head[1].generateNextMove(queue)

while True:
    filename = ""
    print("Pilih input puzzle:")
    print("1. Generate Random Puzzle")
    print("2. Baca dari File")
    option = input("Input: ")
    if option == '1':
        # generate random puzzle
        flatPuzzle = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
        for i in range(0,20):
            randomNum = random.randint(1,12)
            emptyIdx = flatPuzzle.index(16)
            if 1 <= randomNum <= 9 and emptyIdx != 0:
                # move left
                temp = flatPuzzle[emptyIdx-1]
                flatPuzzle[emptyIdx-1] = 16
                flatPuzzle[emptyIdx] = temp
            elif 4 <= randomNum <= 6 and emptyIdx != 15:
                # move right
                temp = flatPuzzle[emptyIdx+1]
                flatPuzzle[emptyIdx+1] = 16
                flatPuzzle[emptyIdx] = temp
            elif 7 <= randomNum <= 9 and not(0 <= emptyIdx <= 3):
                # move up
                temp = flatPuzzle[emptyIdx-4]
                flatPuzzle[emptyIdx-4] = 16
                flatPuzzle[emptyIdx] = temp
            elif 10 <= randomNum <= 12 and not(12 <= emptyIdx <= 15):
                # move down
                temp = flatPuzzle[emptyIdx+4]
                flatPuzzle[emptyIdx+4] = 16
                flatPuzzle[emptyIdx] = temp
        for i in range(16):
            if flatPuzzle[i] == 16:
                flatPuzzle[i] = 'x'
                emptyPos = i+1
        readPuzzle = [[-1 for i in range(4)] for j in range(4)]
        k = 0
        for i in range(4):
            for j in range(4):
                readPuzzle[i][j] = flatPuzzle[k]
```

```
                k += 1
    elif option == '2':
        filename = input("Masukkan nama file (sertakan .txt): ")
        print("Puzzle loaded from:", filename)
        readPuzzle = read.run(filename)

        for i in range(4):
            for j in range(4):
                if readPuzzle[i][j] == 'x':
                    emptyPos = i+1 + 4*j
    else:
        continue

    puzzle = Puzzle(emptyPos)
    for i in range(4):
        for j in range(4):
            if readPuzzle[i][j] == 'x':
                puzzle.elmt[i][j].val = -1
                emptyPos = i+1 + 4*j
            else:
                puzzle.elmt[i][j].val = int(readPuzzle[i][j])

    puzzle.getGoal()

    main(puzzle)

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

e. Instansiasi 5 Buah Persoalan

Nama File	Bentuk Puzzle
input1.txt	5 1 3 4 x 2 6 7 9 10 11 8 13 12 14 15
input2.txt	1 3 7 4 6 2 10 8 5 14 x 11 9 13 15 12
input3.txt	5 1 2 4 9 7 3 8 x 6 10 11 13 14 15 12
input4.txt	1 3 10 4 7 2 14 8 5 x 6 11 9 13 15 12
input5.txt	10 1 x 4 7 5 2 8 13 3 14 12 15 9 11 6

f. Alamat Github

https://github.com/ranjabi/Tucil3_13520002