LAPORAN TUGAS KECIL III

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

Laporan Ini Dibuat Untuk Memenuhi Tugas Perkuliahan

Mata Kuliah Strategi Algoritma (IF2211)

KELAS 02

Dosen : Dr. Nur Ulfa Maulidevi, S.T., M.Sc.

DISUSUN OLEH:

Rava Naufal Attar (13520077)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2021/2022

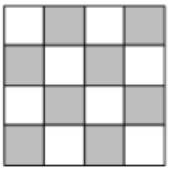
Daftar Isi

Cara Kerja Program

Perlu diketahui, pada program suatu node direpresentasikan sebagai *dictionary* pada bahasa pemrograman `python`. Setiap node memiliki *key* sebagai berikut.

Key	Keterangan
`id`	Menyimpan id node dan bersifat unik
`idBefore`	Menyimpan id parent dari node
`matrix`	Menyimpan matrix puzzle
`fi`	Menyimpan cost f(i) yaitu ongkos dari akar ke simpul i
`gi`	Menyimpan cost g(i) yaitu ongkos dari simpul i ke simpul tujuan
`cost`	Menyimpan cost penjumlahan f(i) dengan g(i)
`lastMove`	Menyimpan gerakan terakhir yang dilakukan oleh ubin kosong pada state node sebelumnya. Direpresentasikan (0 –Up, 1 – Right, 2 – Down, 3 - Left).

Pada matriks puzzle awal yang diterima oleh program dilakukan terlebih dahulu verifikasi untuk mengecek apakah dari matriks awal yang diterima dapat mencapai matriks tujuan dengan menerapkan suatu teorema. Teorema yang dimaksud adalah program hanya dapat mencapai status tujuan apabila penumlahan antara Sigma Kurang(i) dengan X bernilai genap. X akan bernilai 1 apabila sel kosong pada posisi awal terletak pada sel yang diarsir. Jika tidak, maka X akan bernilai 0.



Gambar 1 Arsiran sel 15-puzzle

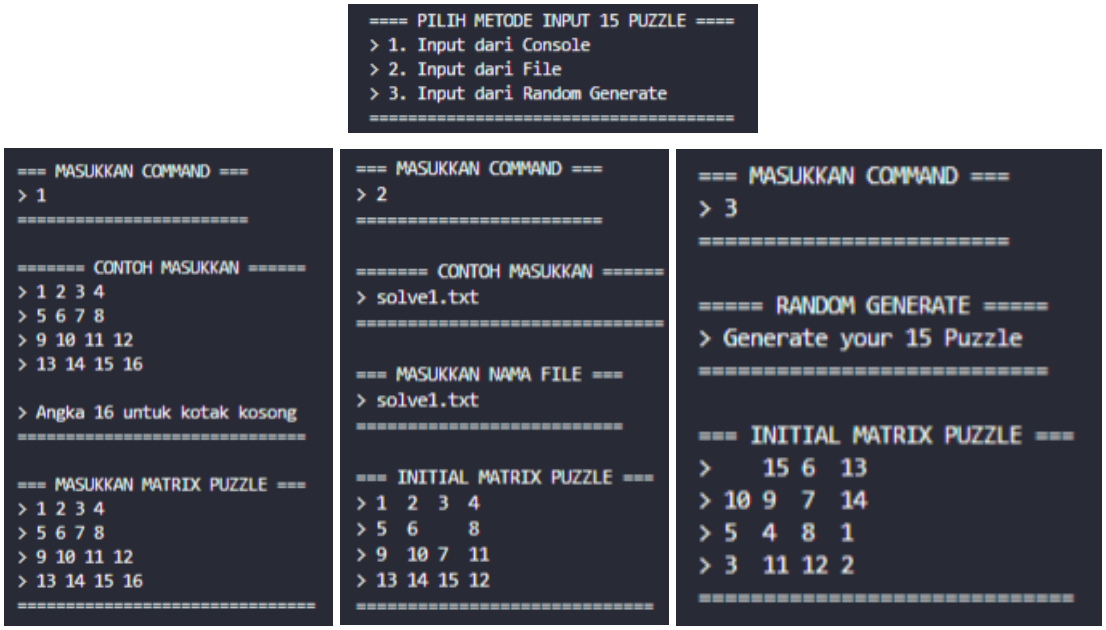
Jika didapatkan hasil bahwa status tujuan dapat dicapai, maka matriks awal akan dimasukkan kedalam antrian simpul hidup. Akan dilakukan ekspansi oleh simpul *expand* yang diambil dari elemen pertama antrian simpul hidup. Saat ekspansi dilakukan, akan terdapat

perhitungan *cost* pada setiap anak dari simpul *expand* yang dapat dihitung sebagaimana telah dijelaskan pada tabel sebelumnya. Kemudian, setiap anak dari simpul *expand* akan dimasukkan kedalam antrian simpul hidup yang diurutkan berdasarkan *cost*-nya. *Cost* termurah akan diletakkan pada sisi depan antrian dan *cost* termahal akan diletakkan pada sisi belakang antrian. Langkah ini dilakukan hingga matriks dari status tujuan tercapai. Program yang dibuat menampung hal-hal sebagai berikut.

Variabel	Keterangan
`simpulE`	Berisi simpul expand
`simpulH`	Antrian dari simpul hidup yang terurut membesar berdasarkan cost-nya
`simpulChecked`	Array yang berisi simpul-simpul yang telah di kunjungi/cek
`pathSimpulFinal`	Array yang berisi simpul-simpul yang mengarahkan dari matriks awal hingga matriks akhir (tujuan)

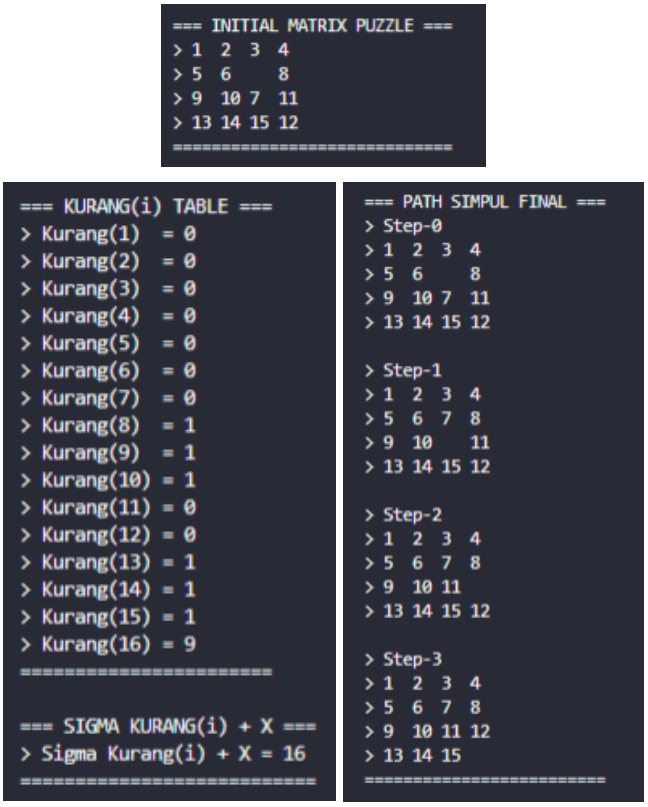
Screenshot Input/Output Program

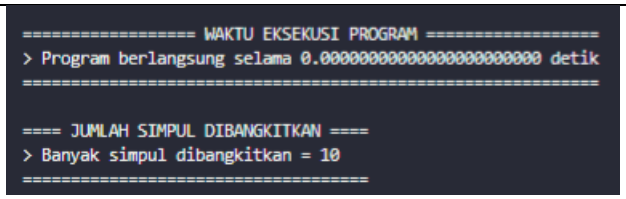
Interface



Gambar 2 Interface

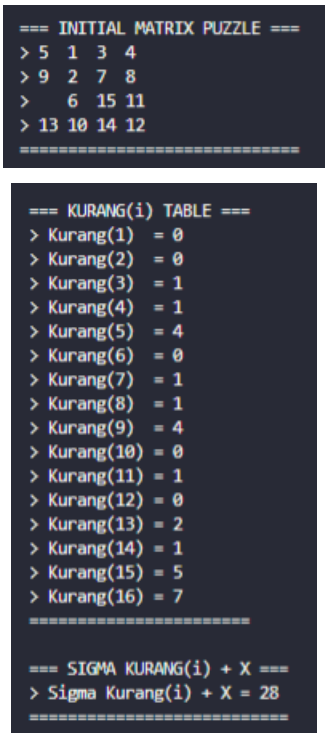
1. Test Case Solve 1

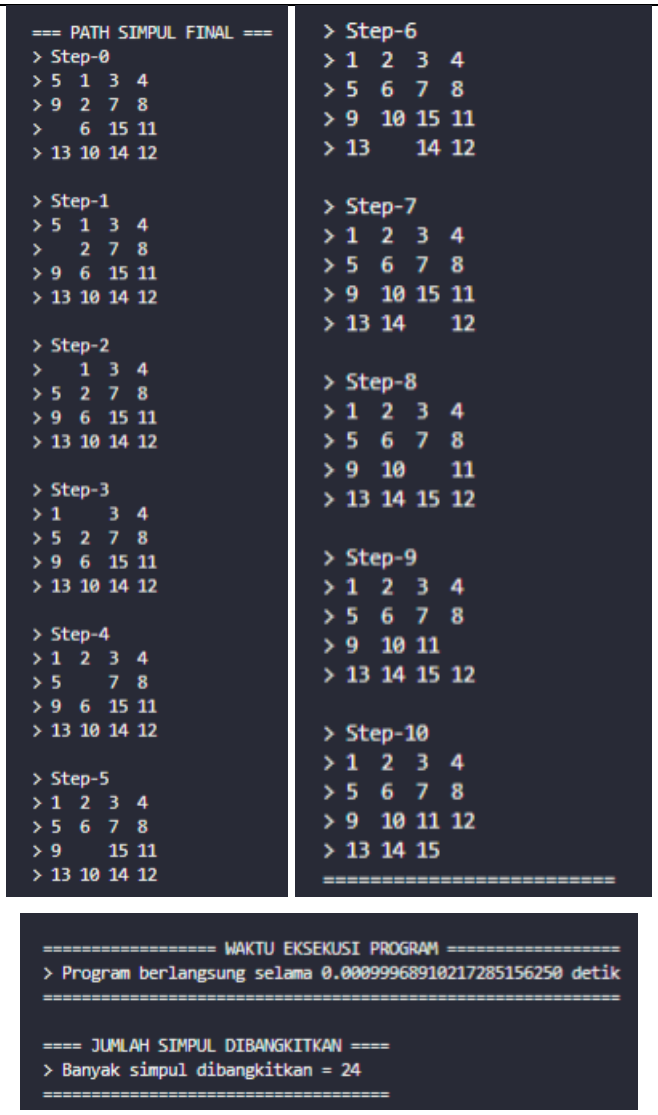




Gambar 2.1 Test case solve 1

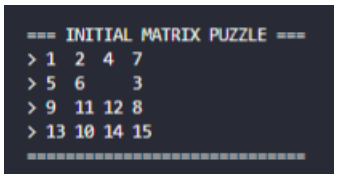
2. Test Case Solve 2





Gambar 2.2 Test case solve 2

3. Test Case Solve 3



```
=== KURANG(i) TABLE ===
> Kurang(1)  = 0
> Kurang(2)  = 0
> Kurang(3)  = 0
> Kurang(4)  = 1
> Kurang(5)  = 1
> Kurang(6)  = 1
> Kurang(7)  = 3
> Kurang(8)  = 0
> Kurang(9)  = 1
> Kurang(10) = 0
> Kurang(11) = 2
> Kurang(12) = 2
> Kurang(13) = 1
> Kurang(14) = 0
> Kurang(15) = 0
> Kurang(16) = 9
======================

=== SIGMA KURANG(i) + X ===
> Sigma Kurang(i) + X = 22
=========================
```

```
=== PATH SIMPUL FINAL ===
> Step-0
> 1  2  4  7
> 5  6     3
> 9  11 12 8
> 13 10 14 15

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

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

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

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

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

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

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

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

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

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

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

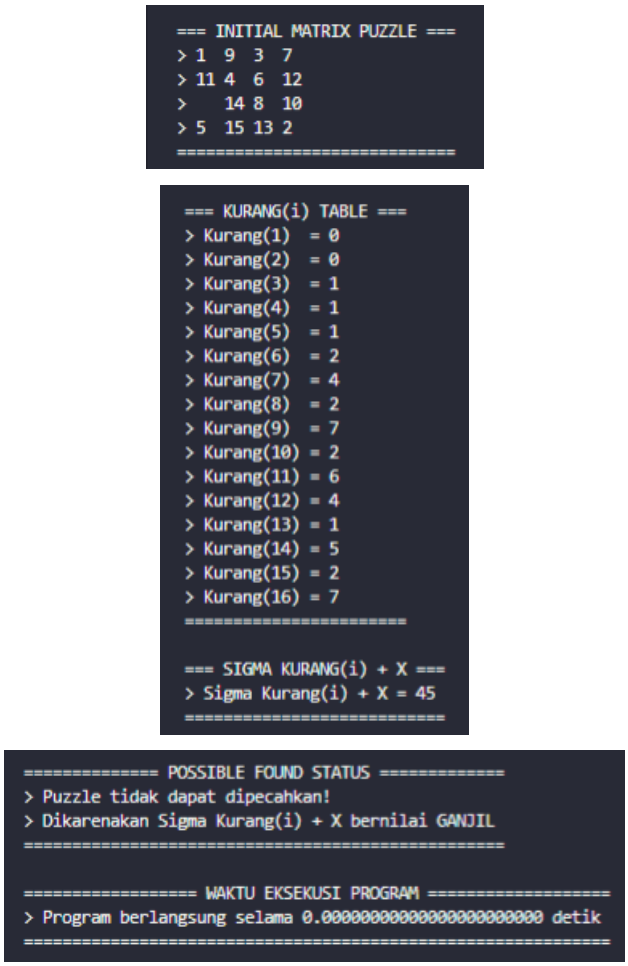
```
================= WAKTU EKSEKUSI PROGRAM =================
> Program berlangsung selama 0.0039899349212646484375O detik
==========================================================

==== JUMLAH SIMPUL DIBANGKITKAN ====
> Banyak simpul dibangkitkan = 71
====================================
```

Gambar 2.3 Test case solve 3

4. Test Case Unsolve 1

Gambar 2.4 Test case unsolve 1

5. Test Case Unsolve 2



Gambar 2.5 Test case unsolve 2

Kode Program

1. InputOutput.py

```python
import numpy as np
from Dependencies import transposeToArray

# INPUT
# input puzzle dari console
def inputFromConsole():
    print("======= CONTOH MASUKKAN ======")
    print("> 1 2 3 4")
    print("> 5 6 7 8")
    print("> 9 10 11 12")
    print("> 13 14 15 16\n")
    print("> Angka 16 untuk kotak kosong")
    print("==============================\n")
    print("=== MASUKKAN MATRIX PUZZLE ===")

    arrTemp = []
    for i in range(4):
        line = input("> ")
        for num in line.split():
            arrTemp.append(int(num))
    print("================================\n")

    return arrTemp

# input puzzle dari file
def inputFromFile():
    print("======= CONTOH MASUKKAN ======")
    print("> solve1.txt")
    print("==============================\n")
    print("=== MASUKKAN NAMA FILE ===")
    namaFile = input("> ")

    arrTemp = []
    with open("./testCase/" + namaFile) as f:
        lines = f.readlines()
    for line in lines:
        for num in line.split():
            arrTemp.append(int(num))
    print("=========================\n")

    return arrTemp

# input puzzle dari random generation
def inputRandomGenerate():
```

```
    print("===== RANDOM GENERATE =====")
    print("> Generate your 15 Puzzle")
    print("==========================\n")

    matrix = np.arange(1, 17)
    np.random.shuffle(matrix)
    matrix = np.reshape(matrix, (4,4))
    arrTemp = transposeToArray(matrix.tolist())

    return arrTemp

# OUTPUT
# menampilkan seluruh hasil fungsi kurang(i)
def printKurangITable(array):
    for i in range(len(array)):
        if(i < 9):
            print(f"> Kurang({i+1})  = {array[i]}")
        else:
            print(f"> Kurang({i+1}) = {array[i]}")

# menampilkan matrix
def printMatrix(matrix):
    for i in range(len(matrix)):
        print("> ", end="")
        for j in range(len(matrix[i])):
            if(matrix[i][j] / 10 < 1):
                print(f"{matrix[i][j]} ", end=" ")
            else:
                if(matrix[i][j] == 16):
                    print("   ", end=" ")
                else:
                    print(f"{matrix[i][j]}", end=" ")
        print()

# menampilkan seluruh node dari matrix awal ke matrix akhir (Goal State)
def printPathSimpulFinal(pathSimpulFinal):
    for i in range(len(pathSimpulFinal)):
        print(f"> Step-{i}")
        printMatrix(pathSimpulFinal[i]["matrix"])

        if(i + 1 != len(pathSimpulFinal)):
            print()
```

2. Dependencies.py

```python
# fungsi untuk transpose dari array ke matrix
def transposeToMatrix(array):
    matrix = []
    baris = []
    for i in range(len(array)):
        baris.append(array[i])
        if ((i + 1) % 4 == 0):
            matrix.append(baris)
            baris = []

    return matrix

# fungsi untuk transpose dari matrix ke array
def transposeToArray(matrix):
    array = []
    for row in matrix:
        for col in row:
            array.append(col)

    return array

# fungsi untuk meng-copy matrix
def copyMatrix(matrix):
    temp = []
    baris = []
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            baris.append(matrix[i][j])
            if (j == len(matrix[i]) - 1):
                temp.append(baris)
                baris = []

    return temp

# fungsi untuk mengecek apakah suatu matrix
# sudah sama dengan Goal state
def isMatrixGoal(matrix):
    finalMatrix =  [
        [1,2,3,4],
        [5,6,7,8],
        [9,10,11,12],
        [13,14,15,16]
    ]
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if (matrix[i][j] != finalMatrix[i][j]):
                return False
```

```
    return True

# fungsi untuk mendapatkan koordinat angka 16 (kotak kosong)
# dari suatu matrix
def getEmptyPosition(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if (matrix[i][j] == 16):
                return [i, j]

# fungsi untuk mendapatkan arah gerakan
# dari kotak kosong yang mungkin dilakukan
def availableMove(node):
    emptyPosition = getEmptyPosition(node["matrix"])
    availMove = []
    # cek move up
    if ((emptyPosition[0] - 1 >= 0) and node["lastMove"] != 2):
        availMove.append(0)
    # cek move right
    if ((emptyPosition[1] + 1 <= 3) and node["lastMove"] != 3):
        availMove.append(1)
    # cek move down
    if ((emptyPosition[0] + 1 <= 3) and node["lastMove"] != 0):
        availMove.append(2)
    # cek move left
    if ((emptyPosition[1] - 1 >= 0) and node["lastMove"] != 1):
        availMove.append(3)

    return availMove

# fungsi untuk mendapatkan state matrix yang
# telah melakukan gerakan tertentu dari state matrix sebelumnya
def move(node, direction, countID):
    newNode = {}
    newNode["id"] = countID
    newNode["idBefore"] = node["id"]
    newNode["matrix"] = []
    newNode["fi"] = node["fi"] + 1
    newNode["gi"] = 0
    newNode["cost"] = node["fi"] + 1
    newMatrix = copyMatrix(node["matrix"])
    emptyPosition = getEmptyPosition(node["matrix"])

    if (direction == 0):
        newMatrix[emptyPosition[0]][emptyPosition[1]] =
node["matrix"][emptyPosition[0] - 1][emptyPosition[1]]
        newMatrix[emptyPosition[0] - 1][emptyPosition[1]] = 16
```

```
        newNode["lastMove"] = 0
    elif (direction == 1):
        newMatrix[emptyPosition[0]][emptyPosition[1]] =
node["matrix"][emptyPosition[0]][emptyPosition[1] + 1]
        newMatrix[emptyPosition[0]][emptyPosition[1] + 1] = 16
        newNode["lastMove"] = 1
    elif (direction == 2):
        newMatrix[emptyPosition[0]][emptyPosition[1]] =
node["matrix"][emptyPosition[0] + 1][emptyPosition[1]]
        newMatrix[emptyPosition[0] + 1][emptyPosition[1]] = 16
        newNode["lastMove"] = 2
    else:
        newMatrix[emptyPosition[0]][emptyPosition[1]] =
node["matrix"][emptyPosition[0]][emptyPosition[1] - 1]
        newMatrix[emptyPosition[0]][emptyPosition[1] - 1] = 16
        newNode["lastMove"] = 3
    newNode["matrix"] = newMatrix

    return newNode

# fungsi untuk menghitung cost g(i) dari suatu state
def calculate_gi(matrix):
    count = 0
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if(matrix[i][j] != ((i * 4) + j + 1) and matrix[i][j] != 16):
                count += 1

    return count

# fungsi untuk mengurutkan collection dari simpul hidup
def sortSimpulHidup(simpulHidup):
    tempSimpulHidup = simpulHidup
    for i in range(0, len(tempSimpulHidup)-1):
        minIdx = i
        for j in range (i+1, len(tempSimpulHidup)):
            if(tempSimpulHidup[j]["cost"] <
tempSimpulHidup[minIdx]["cost"]):
                minIdx = j
        tempVal = tempSimpulHidup[minIdx]
        tempSimpulHidup[minIdx] = tempSimpulHidup[i]
        tempSimpulHidup[i] = tempVal

    return tempSimpulHidup

# fungsi untuk mendapatkan node dengan ID tertentu
# dari node yang telah di cek/kunjungi
def findNode(simpulChecked, nodeID):
```

```
    for node in simpulChecked:
        if node["id"] == nodeID:
            return node

# fungsi untuk mendapatkan seluruh node
# dari matrix awal ke matrix akhir (Goal State)
def getFinalPath(simpulChecked, finalNodeID):
    node = findNode(simpulChecked, finalNodeID)
    temp = []
    temp.append(node)

    while(node["idBefore"] != 0):
        node = findNode(simpulChecked, node["idBefore"])
        temp.append(node)
    temp.reverse()

    return temp
```

3. KurangI.py

```
# fungsi untuk menghitung hasil dari kurang(i)
def kurang_i(i, puzzle):
    i_position = puzzle.index(i)
    sum = 0
    for pos in range(i_position + 1, len(puzzle)):
        if(puzzle[pos] < i):
            sum += 1

    return sum

# fungsi untuk menampung seluruh hasil dari fungsi kurang(i)
def kurang_i_table(puzzle):
    tempArr = [0 for i in range(len(puzzle))]
    for i in range(len(puzzle)):
        kurangIResult = kurang_i(puzzle[i], puzzle)
        tempArr[puzzle[i]-1] = kurangIResult

    return tempArr

# fungsi untuk mendapatkan jumlah dari seluruh hasil fungsi kurang(i)
def sumOf_kurang_i(table):
    sum = 0
    for i in range(len(table)):
        sum += table[i]

    return sum

# fungsi untuk mendapatkan nilai Sigma Kurang(i) + X
```

```python
def kurangI_plusX(puzzle, kurang_i_sum):
    blankPos = puzzle.index(16) + 1
    tempVal = kurang_i_sum
    tempArr = [2,4,5,7,10,12,13,15]
    if (blankPos in tempArr):
        tempVal += 1

    return tempVal
```

4. Main.py

```python
import time

from InputOutput import *
from Dependencies import *
from KurangI import *

# countID untuk menghitung jumlah node yang telah di bangkitkan
countID = 1
# fungsi untuk membangkitkan simpul
def riseNode(node):
    global countID

    moveAvail = availableMove(node)
    simpulHidup = []
    for direction in moveAvail:
        countID = countID + 1
        temp = move(node, direction, countID)
        temp["gi"] = calculate_gi(temp["matrix"])
        temp["cost"] = temp["fi"] + temp["gi"]
        simpulHidup.append(temp)

    return (simpulHidup)

# input angka untuk memilih metode input 15 puzzle
print("\n==== PILIH METODE INPUT 15 PUZZLE ====")
print("> 1. Input dari Console")
print("> 2. Input dari File")
print("> 3. Input dari Random Generate")
print("=======================================\n")

print("=== MASUKKAN COMMAND ===")
inputMethod = input("> ")
print("========================\n")

# validasi input
while(inputMethod not in [1,2,3,'1','2','3']):
    print("=== MASUKKAN TIDAK VALID! ===")
```

```
    print("> PILIH ANGKA 1/2/3!")
    print("============================\n")

    print("=== MASUKKAN COMMAND ===")
    inputMethod = input("> ")
    print("======================\n")

# inisialisasi puzzle sebagai array
initialPuzzleArray = []
if (int(inputMethod) == 1):
    initialPuzzleArray = inputFromConsole()
elif (int(inputMethod) == 2):
    initialPuzzleArray = inputFromFile()
elif (int(inputMethod) == 3):
    initialPuzzleArray = inputRandomGenerate()

timeTakes = 0.0         # inisialisasi waktu eksekusi
startTime = time.time() # waktu dimulai

# beberapa operasi terkait fungsi Kurang(i)
kurangI_table = kurang_i_table(initialPuzzleArray)
kurangI_sum =  sumOf_kurang_i(kurangI_table)
kurangI_plus_x = kurangI_plusX(initialPuzzleArray, kurangI_sum)

timeTakes += (time.time() - startTime)  # menambahkan waktu eksekusi program

print("=== INITIAL MATRIX PUZZLE ===")
printMatrix(transposeToMatrix(initialPuzzleArray))
print("============================\n")

print("=== KURANG(i) TABLE ===")
printKurangITable(kurangI_table)
print("======================\n")

print("=== SIGMA KURANG(i) + X ===")
print(f"> Sigma Kurang(i) + X = {kurangI_plus_x}")
print("==========================\n")

if (kurangI_plus_x % 2 == 0):
    startTime = time.time()     # waktu kembali dimulai

    # inisialisasi node awal puzzle
    initialPuzzle = {
        "id"        : 1,
        "idBefore"  : 0,
        "matrix"    : transposeToMatrix(initialPuzzleArray),
        "fi"        : 0,
        "gi"        : 0,
```

```
        "cost"      : 0,
        "lastMove"  : -1
    }

    # inisialisasi simpulExpand, simpulChecked, pathSimpulFinal, dan
simpulHidup
    simpulE = {}
    simpulChecked = []
    pathSimpulFinal = []
    simpulH = [initialPuzzle]

    while(len(simpulH) != 0):
        simpulE = simpulH.pop(0)
        simpulChecked.append(simpulE)

        # jika matrix sudah sesuai dengan goal, maka keluar dari loop
        if(isMatrixGoal(simpulE["matrix"])):
            simpulE["cost"] = simpulE["fi"]
            pathSimpulFinal = getFinalPath(simpulChecked, simpulE["id"])
            timeTakes += (time.time() - startTime)  # menambahkan waktu
eksekusi program
            break

        # membangkitkan simpul dari simpul terkini
        risedNode = riseNode(simpulE)
        for node in risedNode:
            simpulH.append(node)
        # mengurutkan simpulHidup berdasarkan cost
        simpulH = sortSimpulHidup(simpulH)

    print("=== PATH SIMPUL FINAL ===")
    printPathSimpulFinal(pathSimpulFinal)
    print("==========================\n")

    print("================== WAKTU EKSEKUSI PROGRAM ==================")
    print(f"> Program berlangsung selama {format(timeTakes, '.23f')} detik")
    print("============================================================\n")

    print("==== JUMLAH SIMPUL DIBANGKITKAN ====")
    print(f"> Banyak simpul dibangkitkan = {len(simpulChecked) +
len(simpulH)}")
    print("====================================\n")
else:
    print("============== POSSIBLE FOUND STATUS =============")
    print("> Puzzle tidak dapat dipecahkan!")
    print("> Dikarenakan Sigma Kurang(i) + X bernilai GANJIL")
    print("==================================================\n")
```

```
    print("================== WAKTU EKSEKUSI PROGRAM ====================")
    print(f"> Program berlangsung selama {format(timeTakes, '.23f')} detik")
    print("==============================================================\n")
```

Instansiasi Persoalan

1. testCase/solve1.txt

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

2. testCase/solve2.txt

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

3. testCase/solve3.txt

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

4. testCase/unsolve1.txt

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

5. testCase/unsolve2.txt

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

Alamat Github

https://github.com/sivaren/Stima-Tucil3

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		√