LAPORAN

TUGAS KECIL 3

IF2121 STRATEGI ALGORITMA

Dosen: Dr. Masayu Leylia Khodra, S.T., M.T.

Oleh:

Maharani Ayu Putri Irawan / 13520019

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

BAB I
ALGORITMA BRANCH AND BOUND

Pada tugas kecil ini, digunakan algoritma *branch and bound* dengan deskripsi sebagai berikut:

1. Dibuat node yang menyatakan state awal *puzzle*. Node ini dimasukkan ke dalam sebuah Priority Queue yang mengurutkan node berdasarkan cost (c(i)) minimum.
2. Selama masih ada node yang berada dalam Priority Queue, akan dilakukan pemrosesan sebagai berikut untuk setiap node yang ada:
   a. Diambil node yang memiliki nilai estimasi cost (atau bound) terkecil.
   b. Dicek apakah node tersebut merupakan node solusi. Pengecekan dilakukan dengan mengecek apakah tidak ada tile yang tidak berada pada state yang seharusnya.
   c. Bila node tersebut merupakan node solusi, dilakukan *output*. Pencarian berhenti
   d. Bila node tersebut bukan node solusi, akan dibangkitkan node *child* dengan menggeser *tile* kosong ke seluruh arah yang mungkin di antara atas, kanan, bawah, dan kiri.
   e. Sebuah *child node* yang dibangkitkan perlu dicek validitasnya, yakni apakah *tile* kosong bisa digerakkan ke arah tertentu. Untuk keperluan optimasi, pergerakan *tile* kosong tidak boleh berlawanan dengan pergerakan sebelumnya, serta state *puzzle* yang ada bukan merupakan state node yang pernah dibangkitkan sebelumnya.
   f. Bila node valid dan memenuhi kriteria optimasi, node dimasukkan ke dalam Priority Queue dan state tersebut ditandai agar tidak terdapat pengulangan state yang sama pada pencarian selanjutnya
3. Apabila pencarian berakhir, dilakukan *output*.

BAB II
KODE SUMBER

Kode ditulis dalam Bahasa Python. Berikut merupakan kode sumber yang terdapat di dalam file main.py. Kode sumber juga dapat diakses melalui Github https://github.com/rannnayy/Tucil3_13520019.

```
from library import *

###################### MAIN PROGRAM ############################
welcome()
dict = {}
tempMat = initiate()
tempEmptyTile = [-1, -1]
isSolvable, tableKurang, Sigma, tempEmptyTile[0], tempEmptyTile[1] = checkSolvable(tempMat)

dict[str(toArray(tempMat))] = 0

# conditions if solvable and not
if (isSolvable):
    printMatrix(tempMat)
    printTableKurang(tableKurang)
    print("Sigma Kurang(i) from 1 to 16 + X = " + str(Sigma))
    solve(tempMat, dict, tempEmptyTile)
else:
    print("The puzzle is not solvable")
    printTableKurang(tableKurang)
    print("Sigma Kurang(i) from 1 to 16 + X = " + str(Sigma))
```

Berikut merupakan fungsi-fungsi yang mengimplementasikan algoritma Branch and Bound, yang terdapat pada file library.py.

```
from queue import PriorityQueue
from copy import deepcopy
import random
import time
from node import node
import os
from pathlib import Path

# Output
def printNode(node):
    # For debugging and output purpose
    print(node.noNode)
    printMatrix(node.matrix)
    print()

def printMatrix(matrix):
    # For debugging purpose
    for i in range(4):
        for j in range(4):
            print("%3d"%matrix[i][j], end = " ")
        print()

def printTableKurang(tableKurang):
    # Prints the values of Kurang(i) of all numbers in puzzle, 1-16
    # 16 marks the empty tile
    tableKurang.sort(reverse=False)
    print("Value of each element's Kurang(i):")
    for i in range(len(tableKurang)):
        print("[ " + str("%2d"%tableKurang[i][0]) + " | " + str("%2d"%tableKurang[i][1]) + " ]")

def printPath(node, numStep):
    dictDir = {
```

```
        0: "UP",
        1: "RIGHT",
        2: "DOWN",
        3: "LEFT"
    }
    # print the path from the root node to the goal node
    if (node.parent.parent != None):
        printPath(node.parent, numStep+1)
    else:
        print("Depth : " + str(numStep+1))
    print(dictDir[node.movement], end=" ")
    printNode(node)

def welcome():
    print(" 11  5555     PPPP               l          SSS      l             ")
    print("111  5        P   P              l         S         l             ")
    print(" 11  555  --- PPPP  u  u zz  zz  l eee      SSS  ooo l v v eee rrr ")
    print(" 11     5     P     u  u  z   z  l e e        S o o l v v e e r   ")
    print("1111 555      P      uuu  zz  zz l ee      SSSS  ooo l  v  ee  r   ")
    print()

# Conversion
def toArray(matrix):
    # convert matrix to array
    arr = []
    for i in range(4):
        for j in range(4):
            arr.append(matrix[i][j])
    return arr

# Initialize Puzzle
def initiate():
    # Input puzzle, either from random function or file provided
    choice = int(input("Make a choice!\n1. Input from randomizer\n2. Input from file\nYour choice: "))
    tempMat = []

    if (choice == 1):
        randomData = random.sample(range(1, 17), 16)
        tempMat = [[0 for j in range(4)] for i in range(4)]
        for i in range(4):
            for j in range(4):
                tempMat[i][j] = randomData[i*4 + j]
        printMatrix(tempMat)
    else:
        # Ask for file name input (including extension)
        fileName = input("Enter file name: ")

        # Read from File and Store to Matrix
        currPath = os.path.dirname(__file__)
        relPath = os.path.relpath("Tucil3_13520019\\test\\" + fileName, currPath)
        file = open(Path(relPath), "r")
        lines = file.readlines()

        for line in lines:
            # split to list of integers
            tempMat.append(list(map(lambda x : int(x), line.split())))

    return tempMat

# Initialize Node
def deriveNode(oldMat, oldEmptyTile, movement, level, parent, noNewNode):
    # make a new node with empty tile moved to either up right down or left
    newMat = deepcopy(oldMat)
    # List of additions to directions empty tile movement: up right down left (0,1,2,3)
    add_row = [-1, 0, 1, 0]
    add_col = [0, 1, 0, -1]

    oldX = oldEmptyTile[0]
    oldY = oldEmptyTile[1]

    newX = oldEmptyTile[0] + add_row[movement]
    newY = oldEmptyTile[1] + add_col[movement]

    if (checkIndex((newX, newY))):
```

```
        newMat[oldX][oldY], newMat[newX][newY] = newMat[newX][newY], newMat[oldX][oldY]
        newNode = node(parent, newMat, (newX, newY), noNewNode, movement, level, calcCi(level,
newMat))
        return newNode
    else:
        return None

# Functions
def findEmptyNode(matrix):
    # find the empty tile position
    for i in range(4):
        for j in range(4):
            if (matrix[i][j] == 16):
                return i, j

# Cost estimation
def calcGi(matrix):
    # calculate misplaced tiles
    misplaced = 0
    ctr = 1
    for i in range(4):
        for j in range(4):
            if (matrix[i][j] != ctr):
                misplaced += 1
            ctr += 1
    return misplaced

def calcCi(depth, matrix):
    # calculate cost, with formula c(i) = f(i) + g(i)
    # c(i) = cost
    # f(i) = depth (or level)
    # g(i) = number of misplaced tiles
    return depth + calcGi(matrix)

# Checks
def checkSolvable(mat):
    # check value of Sigma Kurang(i) + X which determines
    # if the puzzle is solvable or not
    tableKurang = []
    totalKurang = 0
    temp = toArray(mat)
    for i in range(0,16):
        kurang = 0
        for j in range(i+1,16):
            if(temp[i]>temp[j]):
                kurang+=1
        tableKurang.append([temp[i], kurang])
        totalKurang += kurang
    i, j = findEmptyNode(mat)
    x = (i + j) % 2
    return ((totalKurang+x) % 2 == 0), tableKurang, totalKurang + x, i, j

def checkPresent(dict, matrix, cost):
    # return true if present in dict and need not to be updated
    if (str(toArray(matrix)) in dict):
        if (dict[str(toArray(matrix))] < cost):
            return True
    return False

def checkIndex(emptyTile):
    # check if movement of empty tile is valid (between 0 and 3 (inclusive)) or not
    return (emptyTile[0] >= 0 and emptyTile[0] < 4 and emptyTile[1] >= 0 and emptyTile[1] < 4)

def noContradictMovement(last, now):
    # return true if not contradicting (the last move was up, on this move, can't go down)
    return ((last == 4) or (last == 0 and now != 2) or (last == 2 and now != 0) or (last == 1 and now
!= 3) or (last == 3 and now != 1))

# Main Function using Branch and Bound Algorithm
def solve(tempMatrix, dict, tempEmptyTile):
    # Solve the N Puzzle using Branch and Bound Algorithm
    start = time.time()

    # make the root node
```

```
    rootNode = node(None, tempMatrix, tempEmptyTile, 1, 4, 0, calcCi(0, tempMatrix))

    # put the root to Priority Queue, sorted from min to max.
    pq = PriorityQueue()
    pq.put((rootNode.ci, rootNode))

    # number of steps taken to reach solution node
    numStep = 0

    # number of node
    ctrNoNode = 1
    while not pq.empty():
        # get first element in Priority Queue (the one with the least cost)
        ci, eNode = pq.get()

        # if reaching the goal state
        if (calcGi(eNode.matrix) == 0):
            end = time.time()
            printPath(eNode, numStep)
            print()
            print("Execution time: " + str(end-start))
            print()
            print("Number of nodes generated: " + str(ctrNoNode))
            break

        # try moves to four directions, if possible
        for i in range(4):
            if (noContradictMovement(eNode.movement, i)):
                childNode = deriveNode(eNode.matrix, eNode.emptyTile, i, eNode.fi + 1, eNode,
ctrNoNode + 1)

                if (childNode != None and not checkPresent(dict, childNode.matrix, childNode.ci)):
                    # put new node to Priority Queue and Dictionary
                    pq.put((childNode.ci, childNode))
                    dict[str(toArray(childNode.matrix))] = childNode.ci
                    ctrNoNode += 1
```

Berikut merupakan struktur data node yang dipakai untuk menyimpan state puzzle pada suatu langkah. Kode berikut terdapat pada file node.py

```
# Node to store the puzzle state at a time
class node:
    def __init__(self, parent, matrix, emptyTile, noNode, movement, level, cost):
        # Parent Node
        self.parent = parent
        # Puzzle state in matrix
        self.matrix = matrix
        # Tuple (row, column) of the empty tile's location
        self.emptyTile = emptyTile
        # Number of node
        self.noNode = noNode
        # Movement that generates this node from the parent
        self.movement = movement
        # Level of the node
        self.fi = level
        # Cost
        self.ci = cost

    def __lt__(self, other):
        # Comparation done by cost
        return self.ci < other.ci
```

BAB III

EKSPERIMEN

A. Masukan Program

```
 11  5555     PPPP              l          SSS      l
111  5        P   P             l         S         l
 11  555  --- PPPP  u  u zz  zz  l eee      SSS  ooo l v v eee rrr
 11     5     P     u  u  z   z  l e e        S o o l v v e e r
1111 555      P      uuu  zz  zz l ee      SSSS  ooo l  v  ee  r

Make a choice!
1. Input from randomizer
2. Input from file
Your choice:
```

B. Kasus Uji

Dalam kasus uji yang dibuat untuk masing-masing langkah, kasus tersebut tidak dibuat untuk diselesaikan dengan tepat N langkah, melainkan sekitar N langkah (perkiraan).

1. Input dari randomizer

```
Your choice: 1
  6   9   2   3
  5  14   1  10
 15   7  16  13
  4  12   8  11
The puzzle is not solvable
Value of each element's Kurang(i):
[  1 |  0 ]
[  2 |  1 ]
[  3 |  1 ]
[  4 |  0 ]
[  5 |  2 ]
[  6 |  5 ]
[  7 |  1 ]
[  8 |  0 ]
[  9 |  7 ]
[ 10 |  3 ]
[ 11 |  0 ]
[ 12 |  2 ]
[ 13 |  4 ]
[ 14 |  8 ]
[ 15 |  6 ]
[ 16 |  5 ]
Sigma Kurang(i) from 1 to 16 + X = 45
```

2. Input dari file, kedalaman solusi 5

```
Your choice: 2
Enter file name: 5step.txt
  1   2   3   4
  5  10   6   8
  9  16   7  12
 13  14  11  15
Value of each element's Kurang(i):
[  1 |  0 ]
[  2 |  0 ]
[  3 |  0 ]
[  4 |  0 ]
[  5 |  0 ]
[  6 |  0 ]
[  7 |  0 ]
[  8 |  1 ]
[  9 |  1 ]
[ 10 |  4 ]
[ 11 |  0 ]
[ 12 |  1 ]
[ 13 |  1 ]
[ 14 |  1 ]
[ 15 |  0 ]
[ 16 |  6 ]
Sigma Kurang(i) from 1 to 16 + X = 16
Depth : 5
UP 2
  1   2   3   4
  5  16   6   8
  9  10   7  12
 13  14  11  15
RIGHT 7
  1   2   3   4
  5   6  16   8
  9  10   7  12
 13  14  11  15

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

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

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


Execution time: 0.0

Number of nodes generated: 16
```

3. Input dari file, kedalaman solusi 10

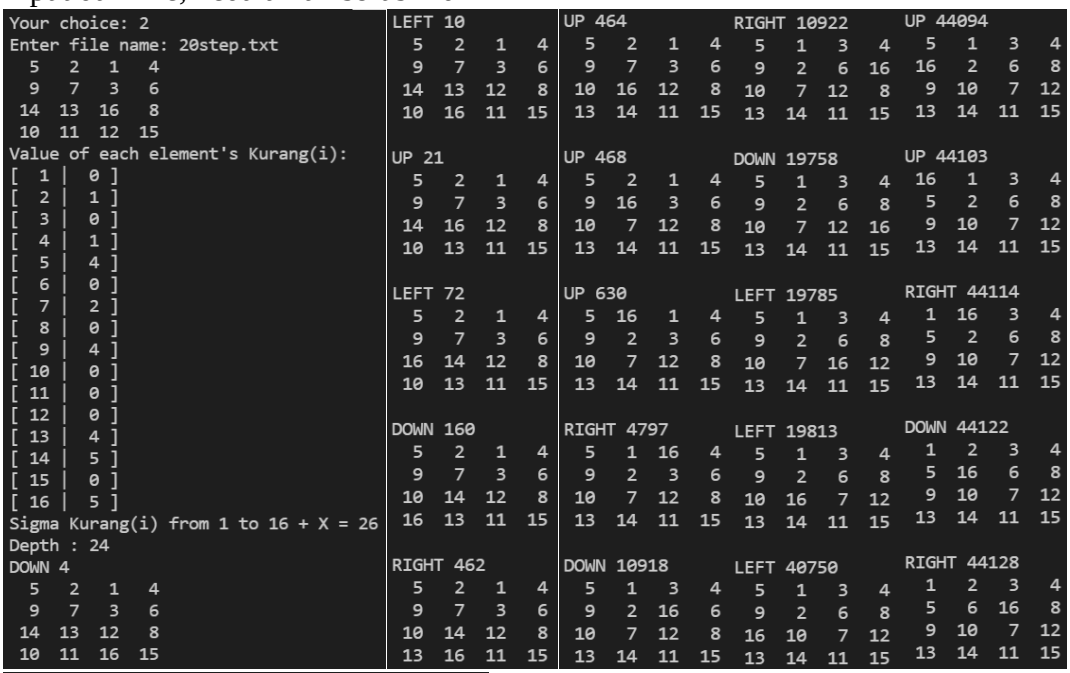
```
Your choice: 2
Enter file name: 10step.txt
  5   1   3   4
  9   2  11   7
 16   6  10   8
 13  14  15  12
Value of each element's Kurang(i):
[  1 |  0 ]
[  2 |  0 ]
[  3 |  1 ]
[  4 |  1 ]
[  5 |  4 ]
[  6 |  0 ]
[  7 |  1 ]
[  8 |  0 ]
[  9 |  4 ]
[ 10 |  1 ]
[ 11 |  4 ]
[ 12 |  0 ]
[ 13 |  1 ]
[ 14 |  1 ]
[ 15 |  1 ]
[ 16 |  7 ]
Sigma Kurang(i) from 1 to 16 + X = 26
Depth : 10
UP 2
  5   1   3   4
 16   2  11   7
  9   6  10   8
 13  14  15  12
UP 5
 16   1   3   4
  5   2  11   7
  9   6  10   8
 13  14  15  12

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

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

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

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

RIGHT 20
  1   2   3   4
  5   6   7  16
  9  10  11   8
 13  14  15  12

DOWN 23
  1   2   3   4
  5   6   7   8
  9  10  11  16
 13  14  15  12

DOWN 24
  1   2   3   4
  5   6   7   8
  9  10  11  12
 13  14  15  16

Execution time: 0.0010020732879638672

Number of nodes generated: 25
```

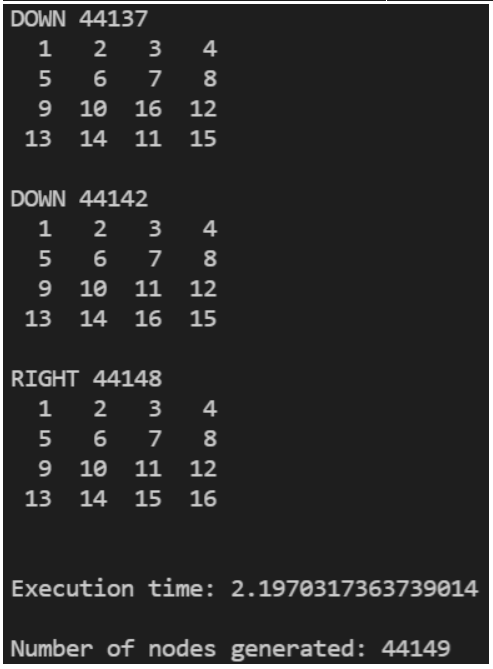
4. Input dari file, kedalaman solusi 15

```
Your choice: 2
Enter file name: 15step.txt
 16   5   2   4
  1   9   3   7
  6  11  12   8
 13  10  14  15
Value of each element's Kurang(i):
[  1 |  0 ]
[  2 |  1 ]
[  3 |  0 ]
[  4 |  2 ]
[  5 |  4 ]
[  6 |  0 ]
[  7 |  1 ]
[  8 |  0 ]
[  9 |  4 ]
[ 10 |  0 ]
[ 11 |  2 ]
[ 12 |  2 ]
[ 13 |  1 ]
[ 14 |  0 ]
[ 15 |  0 ]
[ 16 | 15 ]
Sigma Kurang(i) from 1 to 16 + X = 32
Depth : 16
DOWN 3
  1   5   2   4
 16   9   3   7
  6  11  12   8
 13  10  14  15
RIGHT 4
  1   5   2   4
  9  16   3   7
  6  11  12   8
 13  10  14  15

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

RIGHT 70
  1   2  16   4
  9   5   3   7
  6  11  12   8
 13  10  14  15

DOWN 80
  1   2   3   4
  9   5  16   7
  6  11  12   8
 13  10  14  15

RIGHT 86
  1   2   3   4
  9   5   7  16
  6  11  12   8
 13  10  14  15
DOWN 94
  1   2   3   4
  9   5   7   8
  6  11  12  16
 13  10  14  15

LEFT 99
  1   2   3   4
  9   5   7   8
  6  11  16  12
 13  10  14  15

LEFT 105
  1   2   3   4
  9   5   7   8
  6  16  11  12
 13  10  14  15

LEFT 111
  1   2   3   4
  9   5   7   8
 16   6  11  12
 13  10  14  15

UP 122
  1   2   3   4
 16   5   7   8
  9   6  11  12
 13  10  14  15
RIGHT 264
  1   2   3   4
  5  16   7   8
  9   6  11  12
 13  10  14  15

DOWN 273
  1   2   3   4
  5   6   7   8
  9  16  11  12
 13  10  14  15

DOWN 275
  1   2   3   4
  5   6   7   8
  9  10  11  12
 13  16  14  15

RIGHT 277
  1   2   3   4
  5   6   7   8
  9  10  11  12
 13  14  16  15

RIGHT 280
  1   2   3   4
  5   6   7   8
  9  10  11  12
 13  14  15  16
```

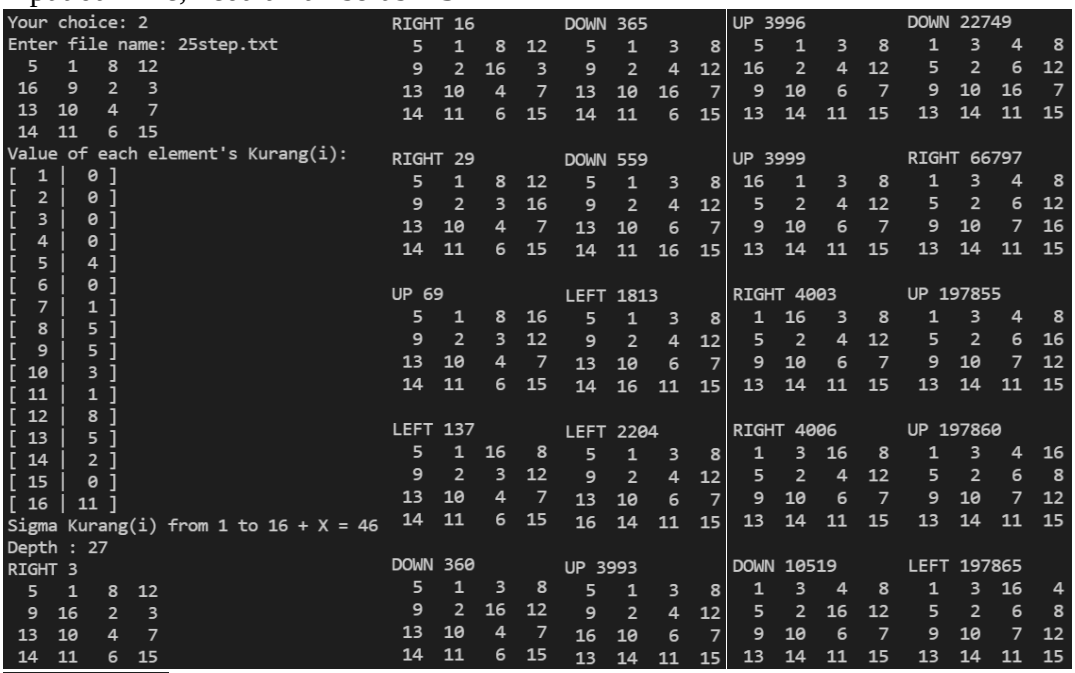
```
Execution time: 0.009011268615722656

Number of nodes generated: 280
```

5. Input dari file, kedalaman solusi 20





6. Input dari file, kedalaman solusi 25



7. Input dari file, tidak ditemukan solusi

```
Your choice: 2
Enter file name: fail.txt
The puzzle is not solvable
Value of each element's Kurang(i):
[  1 |  0 ]
[  2 |  0 ]
[  3 |  2 ]
[  4 |  0 ]
[  5 |  2 ]
[  6 |  2 ]
[  7 |  6 ]
[  8 |  1 ]
[  9 |  8 ]
[ 10 |  5 ]
[ 11 |  4 ]
[ 12 |  1 ]
[ 13 |  1 ]
[ 14 |  3 ]
[ 15 |  2 ]
[ 16 | 10 ]
Sigma Kurang(i) from 1 to 16 + X = 47
```

BAB IV

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		✓