LAPORAN TUGAS KECIL 3

IF2211 Strategi Algoritma

Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound

Disusun oleh:

Nama : Gede Sumerta Yoga

NIM : 13520021

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

Algoritma Branch & Bounds

Dalam penerapan tugas kecil 3 Strategi Algoritma ini, algoritma yang harus digunakan adalah algoritma Branch & Bound. Algoritma ini digunakan untuk menyelesaikan persoalan optimasi. Branch & Bound ini bisa dibilang merupakan penggabungan dari BFS dan least cost search. Pada algoritma ini, setiap simpul diberi sebuah nilai cost dan pemilihan simpul yang akan di-expand dipilih dari nilai cost setiap simpul

Dalam tugas ini, permasalahan yang diangkat adalah menyelesaikan persoalan 15-Puzzle. Perhitungan nilai cost dari setiap simpul yang digunakan adalah penjumlahan dari jarak simpul akar ke simpul tersebut dengan banyaknya nomor ubin yang tidak sesuai posisi seharusnya di puzzle. Untuk menyelesaikan tugas ini saya membuat sebuah file puzzleSolver.py yang berisi kumpulan fungsi untuk menyelesaikan permasalahan 15-puzzle ini. Berikut penjelasan dari setiap fungsi:

Fungsi	Keterangan
fillPuzzleFromList(puzzle, list)	Mengisi setiap elemen puzzle dengan elemen yang ada di list
printPuzzle(matrix, file)	Mencetak matrix/puzzle ke file dan terminal
posisiKurang(matrix, bilPertama, bilKedua)	Mengecek apakah posisi bilPertama pada puzzle kurang dari posisi bilKedua pada puzzle
posisi(matrix, x)	Mencari posisi baris dan kolom nilai x di puzzle/matrix
posisiSelKosong(matrix)	Mengecek posisi sel kosong di matrix untuk menentukan nilainya di penentuan cost simpul
Kurang(matrix,x)	Menghitung nilai fungsi Kurang dari nilai x pada puzzle
sigmaKurang(matrix)	Menghitung total nilai fungsi Kurang dari setiap nilai pada puzzle
puzzleCanBeSolve(total)	Mengecek apakah puzzle bisa diselesaikan berdasarkan nilai total yaitu sigmaKurang+X

ubinSalahPosisi(matrix)	Menghitung banyak elemen pada puzzle yang berada tidak pada posisi seharusnya
pindahUbin(matrix, posisiBaris, posisiKolom, arahVertikal, arahHorizontal)	Menukar nilai elemen pada posisiBaris dan posisiKolom pada puzzle dengan memperhatikan arahVertikal dan arahHorizontal
pindahkanSlotKosong(matriks, arah)	Memindahkan slot kosong sesuai masukan arah
isGoal(matrix)	Mengecek apakah matrix/puzzle merupakan tujuan
findId(listSimpul, id)	Mencari simpul id di listSimpul
findLangkah(listSimpul, id)	Mencari rute dari root ke simpul tujuan yang memiliki ID id

Pada tugas kecil kali ini, saya membuat dua versi penyelesaian yaitu dengan menggunakan command line di file main.py dan menggunakan GUI di file GUI.py. Cara kerja keduanya cukup mirip dengan menggunakan fungsi-fungsi di puzzleSolver.py. Secara umum cara kerja program saya adalah seperti berikut:

1. Menerima input puzzle dari file dengan memasukkan nama file yang ada di folder test atau dengan me-random puzzle
2. Setiap elemen pada puzzle dihitung nilainya terhadap fungsi Kurang
3. Menjumlahkan nilai fungsi Kurang dari setiap elemen dan posisi sel kosong
4. Mempertimbangkan apakah puzzle bisa diselesaikan berdasarkan nilai pada nomor 3
5. Jika puzzle bisa diselesaikan, lanjutan ada di nomor 7
6. Jika puzzle tidak bisa diselesaikan, mencetak pesan.
7. Memasukkan simpul awal ke dalam Priority Queue
8. Mengambil elemen pertama pada Priority Queue
9. Meng-expand simpul yang terpilih menjadi maksimal 4 simpul yang merupakan puzzle baru dengan pergeseran ubin kosong. Simpul dibangkitkan jika bukan merupakan perulangan dari orangtua simpul yang di-expand dan slot kosong bisa dipindahkan sesuai arah masukan. Setiap simpul yang dibangkitkan dicatat di listSimpul
10. Mengecek simpul yang dibangkitkan apakah merupakan simpul tujuan atau bukan. Jika bukan simpul tujuan ulangi dari langkah ke-8.

11. Jika merupakan simpul tujuan, dicari rute dari simpul awal ke simpul tujuan dan menampilkan setiap langkah yang dilalui

Setiap bagian yang dicetak ke command line akan tercetak juga di file output.txt, seperti puzze awal, nilai fungsi Kurang, dan langkah penyelesaian. Pada penerimaan file input, konfigurasi elemen dalam file dibentuk seperti matrix dengan slot kosong diganti angka 16, tetapi pada puzzle tetap dicetak slot kosong.

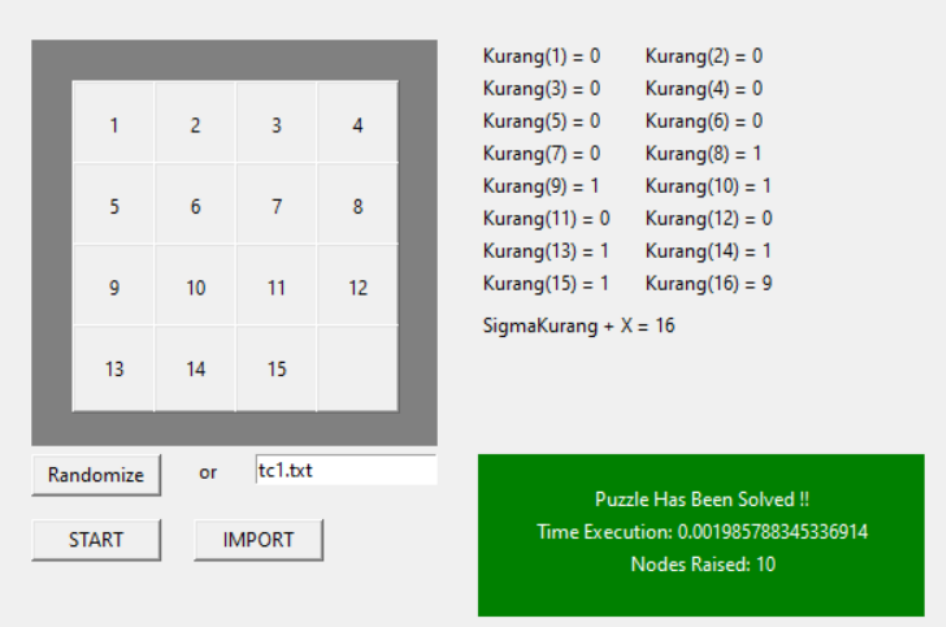
Input & Output Program

1. tc1.txt

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

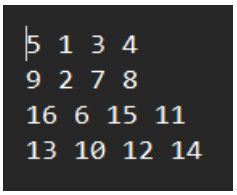
Gambar 1. tc1.txt

Gambar 2. Puzzle awal tc1.txt



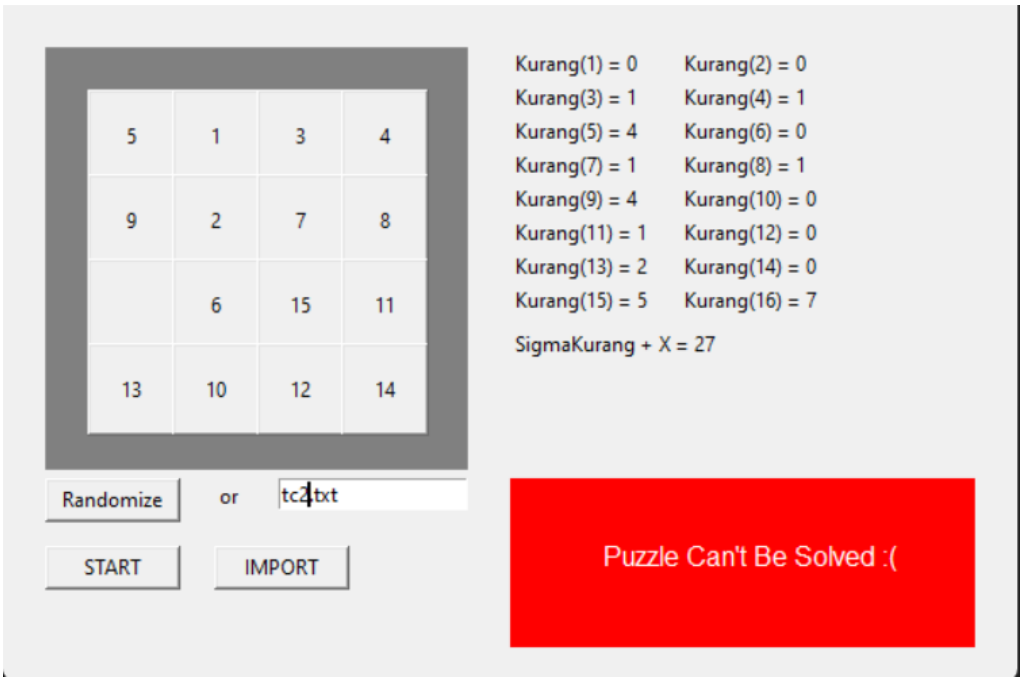
Gambar 3. Hasil akhir tc1.txt

2. tc2.txt



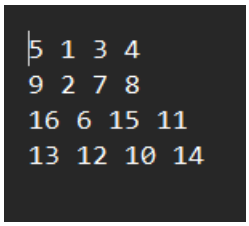
Gambar 4. tc2.txt

Gambar 5. Puzzle awal tc2.txt

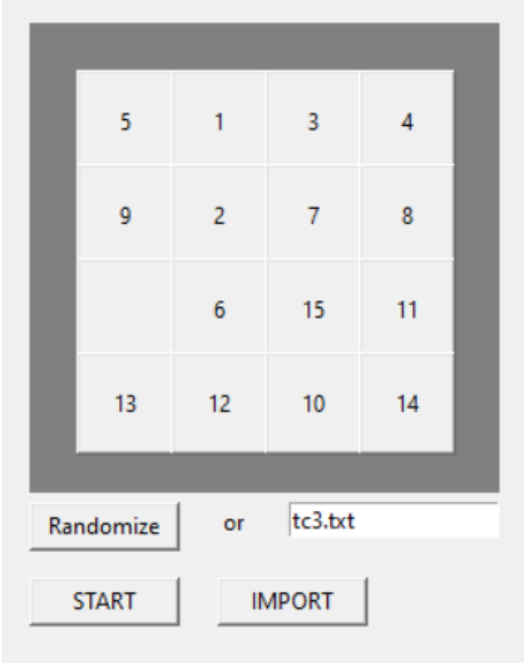


Gambar 6. Hasil akhir tc2.txt

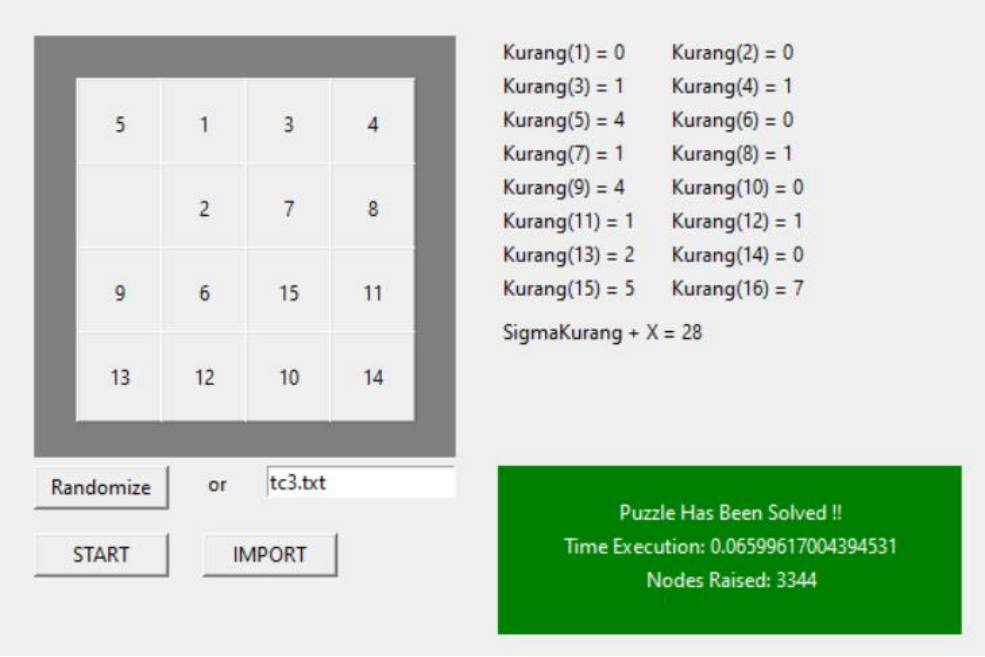
3. tc3.txt



Gambar 7. tc3.txt

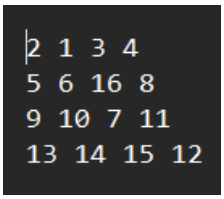


Gambar 8. Puzzle awal tc3.txt

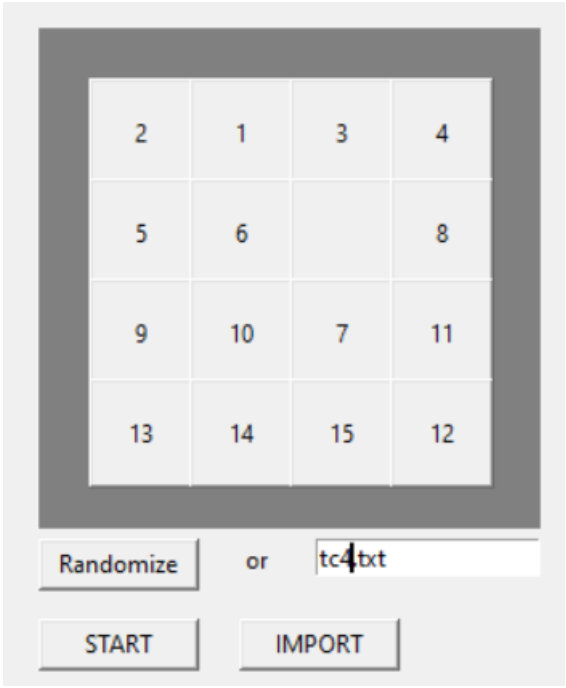


Gambar 9. Hasil akhir tc3.txt

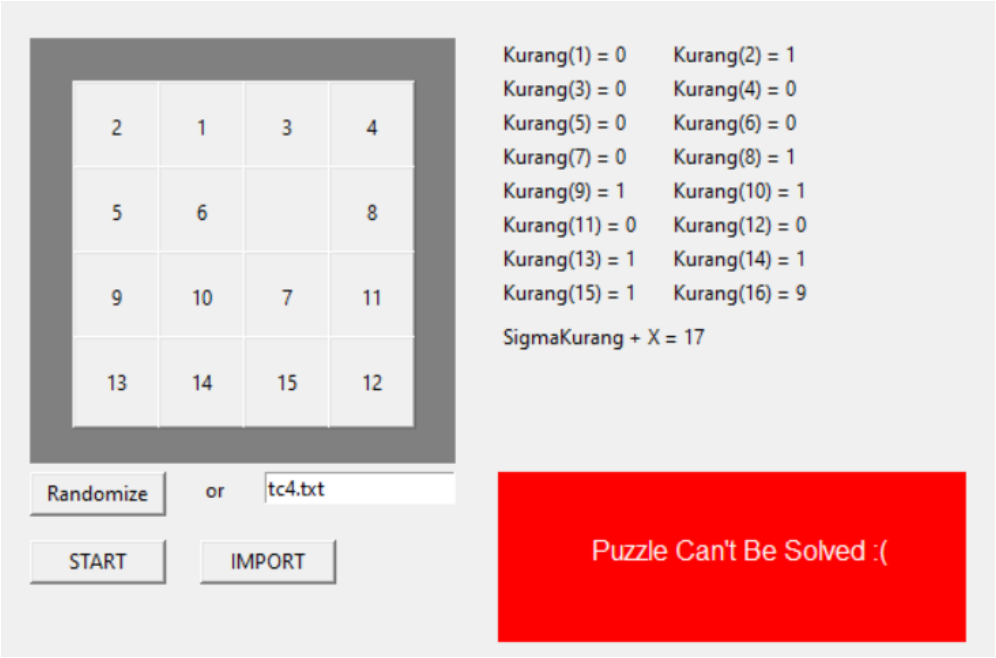
4. tc4.txt



Gambar 10. tc5.txt



Gambar 11. Puzzle awal tc4.txt



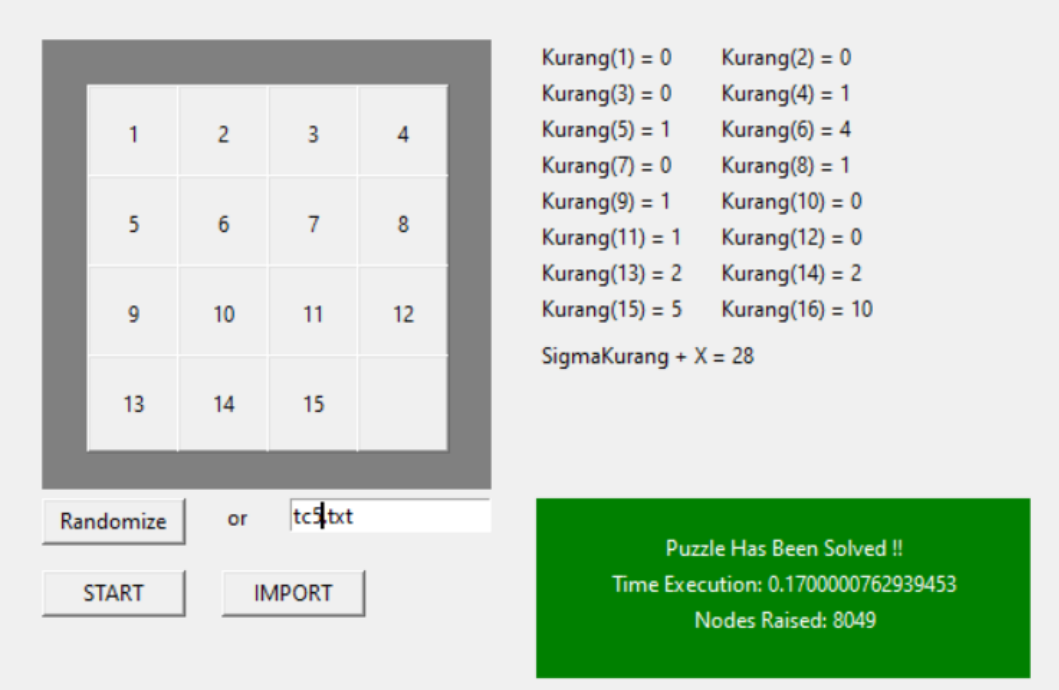
Gambar 12. Hasil akhir tc4.txt

5. tc5.txt

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

Gambar 13. tc5.txt

Gambar 14. Puzzle awal tc5.txt



Gambar 15. Hasil akhir tc5.txt

Lampiran

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	✓	

puzzleSolver.py

```
from copy import deepcopy

# File ini berisi kumpulan fungsi
# untuk menyelesaikan permasalahan
# 15 Puzzle


def fillPuzzleFromList(puzzle, list):
    iter = 0
    for i in range(4):
        for j in range(4):
            puzzle[i][j] = list[iter]
            iter += 1


def printPuzzle(matrix, file):
    for i in range(4):
        for j in range(4):
            if(matrix[i][j] == 16):
                print("   -", end=" ")
                file.write("   -")
            else:
                print('{:4}'.format(matrix[i][j]), end=" ")
                file.write('{:4}'.format(matrix[i][j]))
        print()
        file.write("\n")


def posisiKurang(matrix, bilPertama, bilKedua):
    barisBilPertama, kolomBilPertama = posisi(matrix, bilPertama)
    barisBilKedua, kolomBilKedua = posisi(matrix, bilKedua)
    return barisBilPertama*4+kolomBilPertama < barisBilKedua*4+kolomBilKedua


def posisi(matrix, x):
    i = 0
    while(i < 4):
        j = 0
        while(j < 4):
            if(matrix[i][j] == x):
                return i, j
            j += 1
        i += 1


def posisiSelKosong(matrix):
    i, j = posisi(matrix, 16)
```

```
    if((i+j) % 2 == 0):
        return 0
    else:
        return 1


def Kurang(matrix, x):
    count = 0
    for i in range(4):
        for j in range(4):
            if((matrix[i][j] < x) and posisiKurang(matrix, x, matrix[i][j])):
                count += 1
    return count


def sigmaKurang(matrix):
    count = 0
    for i in range(4):
        for j in range(4):
            count += Kurang(matrix, matrix[i][j])
    return count


def puzzleCanBeSolve(matrix, total):
    return total % 2 == 0


def ubinSalahPosisi(matrix):
    count = 0
    for i in range(4):
        for j in range(4):
            if(matrix[i][j] != i*4+j+1 and matrix[i][j] != 16):
                count += 1
    return count


def pindahUbin(matrix, posisiBaris, posisiKolom, arahVertikal,
arahHorizontal):
    temp = matrix[posisiBaris][posisiKolom]
    matrix[posisiBaris][posisiKolom] = matrix[posisiBaris +
                                              arahVertikal][posisiKolom +
arahHorizontal]
    matrix[posisiBaris + arahVertikal][posisiKolom + arahHorizontal] = temp


def pindahkanSlotKosong(matriks, arah):
    matrix = deepcopy(matriks)
    barisSelKosong, kolomSelKosong = posisi(matrix, 16)
    if(arah == 0):  # Up
```

```
        if(barisSelKosong != 0):
            pindahUbin(matrix, barisSelKosong, kolomSelKosong, -1, 0)
    elif(arah == 1):  # Right
        if(kolomSelKosong != 3):
            pindahUbin(matrix, barisSelKosong, kolomSelKosong, 0, 1)
    elif(arah == 2):  # Down
        if(barisSelKosong != 3):
            pindahUbin(matrix, barisSelKosong, kolomSelKosong, 1, 0)
    elif(arah == 3):  # Left
        if(kolomSelKosong != 0):
            pindahUbin(matrix, barisSelKosong, kolomSelKosong, 0, -1)
    return matrix


def isGoal(matrix):
    for i in range(4):
        for j in range(4):
            if(matrix[i][j] != i*4+j+1):
                return False
    return True


def findId(listSimpul, id):
    for i in range(len(listSimpul)):
        if (listSimpul[i][0] == id):
            return listSimpul[i]


def findLangkah(listSimpul, id):
    listLangkah = []
    while(id != 1):
        temp = findId(listSimpul, id)
        listLangkah.insert(0, temp[2])
        id = temp[1]
    return listLangkah
```

main.py

```
import heapq
import time
from puzzleSolver import *
import random

masukanTidakSesuai = True
while(masukanTidakSesuai):
    print("Pilih cara membentuk puzzle")
    print("1. Random")
    print("2. Baca File")
    pilihan = int(input("Ketik pilihan (1/2): "))
    puzzle = [[0 for j in range(4)] for i in range(4)]
    if(pilihan == 1):
        randomList = random.sample(range(1, 17), 16)
        fillPuzzleFromList(puzzle, randomList)
        outputFile = open("output.txt", "w")
        masukanTidakSesuai = False
    elif(pilihan == 2):
        fileName = input("\nMasukkan nama file: ")
        file = open("./test/" + fileName, "r")
        outputFile = open("./test/output.txt", "w")
        for i in range(4):
            f = file.readline().split()
            for j in range(4):
                puzzle[i][j] = int(f[j])
        file.close()
        masukanTidakSesuai = False
    else:
        print("Masukan tidak sesuai. Ulangi!")

print("Puzzle: ")
outputFile.write("Puzzle: \n")
printPuzzle(puzzle, outputFile)
print()
outputFile.write("\n")

start = time.time()
# Penghitungan dan Pencetakan nilai Kurang(i)
for i in range(1, 17):
    print("Kurang(" + str(i) + ") = " + str(Kurang(puzzle, i)))
    outputFile.write("Kurang(" + str(i) + ") = " + str(Kurang(puzzle, i)))
    outputFile.write("\n")

totalValue = sigmaKurang(puzzle) + posisiSelKosong(puzzle)
print("\nSigmaKurang + X = " + str(totalValue))
outputFile.write("\nSigmaKurang + X = " + str(totalValue) + "\n")
```

```
if(puzzleCanBeSolve(totalValue)):
    print("Puzzle bisa diselesaikan")
    outputFile.write("Puzzle bisa diselesaikan\n")

    prioQueue = []
    listSimpul = []
    simpulChecked = 1
    found = False
    heapq.heappush(prioQueue, (0, 0, 0, puzzle, 1, 0))
    while(len(prioQueue) != 0 and not found):
        simpul = heapq.heappop(prioQueue)
        # simpul struct:
        # 1. prio
        # 2. arah
        # 3. depth
        # 4. matrix
        # 5. id
        # 6. parentid
        for i in range(4):
            if((i+2) % 4 == simpul[1] and simpul[2] != 0):
                continue
            puzzleMove = pindahkanSlotKosong(simpul[3], i)
            if(puzzleMove == simpul[3]):
                continue
            # print(simpulChecked)
            simpulChecked += 1
            if(isGoal(puzzleMove)):
                listSimpul.append((simpulChecked, simpul[4], puzzleMove))
                prio = simpul[2]+1 + ubinSalahPosisi(puzzleMove)
                found = True
                break
            else:
                heapq.heappush(prioQueue, (simpul[2]+1+ubinSalahPosisi(
                    puzzleMove), i, simpul[2]+1, puzzleMove, simpulChecked,
simpul[4]))
                listSimpul.append((simpulChecked, simpul[4], puzzleMove))
    end = time.time()
    idGoal = simpulChecked
    iter = 1
    listLangkah = findLangkah(listSimpul, idGoal)
    for i in listLangkah:
        print("Langkah ke-" + str(iter))
        outputFile.write("Langkah ke-" + str(iter) + "\n")
        printPuzzle(i, outputFile)
        print()
        outputFile.write("\n")
        iter += 1
```

```
    print("Jumlah simpul yang dibangkitkan: " +
          str(simpulChecked))
    print(f"Runtime of the program is {end - start}")
    outputFile.write("Jumlah simpul yang dibangkitkan: " +
                     str(simpulChecked) + "\n")
    outputFile.write(f"Runtime of the program is {end - start}")
else:
    print("Puzzle tidak bisa diselesaikan")
    outputFile.write("Puzzle tidak bisa diselesaikan ")
outputFile.close()
```

GUI.py

```
import heapq
import tkinter
import time
import random
from turtle import width
from puzzleSolver import *


def startSolver():
    outputFile = open("../test/output.txt", "w")
    outputFile.write("Puzzle: \n")
    printPuzzle(puzzle, outputFile)

    start = time.time()
    listKurangLabel = []
    for i in range(0, 8):
        listKurangLabel.append(tkinter.Label(
            root, text="Kurang(" + str(i*2+1) + ") = " + str(Kurang(puzzle,
i*2+1))))
        listKurangLabel[i*2].place(x=300, y=24+(i)*20)
        listKurangLabel.append(tkinter.Label(
            root, text="Kurang(" + str(i*2+2) + ") = " + str(Kurang(puzzle,
i*2+2))))
        listKurangLabel[i*2+1].place(x=400, y=24+(i)*20)
        outputFile.write("Kurang(" + str(i*2+1) + ") = " +
                         str(Kurang(puzzle, i*2+1)) + "\n")
        outputFile.write("Kurang(" + str(i*2+2) + ") = " +
                         str(Kurang(puzzle, i*2+2)) + "\n")

    totalValue = sigmaKurang(puzzle) + posisiSelKosong(puzzle)
    sigmaKurangLabel = tkinter.Label(
```

```
        root, text="SigmaKurang + X = " + str(totalValue))
    sigmaKurangLabel.place(x=300, y=190)
    outputFile.write("SigmaKurang + X = " + str(totalValue) + "\n")

    if(puzzleCanBeSolve(totalValue)):
        prioQueue = []
        listSimpul = []
        simpulChecked = 1
        found = False
        heapq.heappush(prioQueue, (0, 0, 0, puzzle, 1, 0))
        while(len(prioQueue) != 0 and not found):
            simpul = heapq.heappop(prioQueue)
            # simpul struct:
            # 1. prio
            # 2. arah
            # 3. depth
            # 4. matrix
            # 5. id
            # 6. parentid
            for i in range(4):
                if((i+2) % 4 == simpul[1] and simpul[2] != 0):
                    continue
                puzzleMove = pindahkanSlotKosong(simpul[3], i)
                if(puzzleMove == simpul[3]):
                    continue
                # print(simpulChecked)
                simpulChecked += 1
                if(isGoal(puzzleMove)):
                    listSimpul.append((simpulChecked, simpul[4], puzzleMove))
                    prio = simpul[2]+1 + ubinSalahPosisi(puzzleMove)
                    found = True
                    break
                else:
                    heapq.heappush(prioQueue, (simpul[2]+1+ubinSalahPosisi(
                        puzzleMove), i, simpul[2]+1, puzzleMove,
simpulChecked, simpul[4]))
                    listSimpul.append((simpulChecked, simpul[4], puzzleMove))
        end = time.time()
        finishFrame = tkinter.Frame(master=root, width=275,
                                    height=100,
background="green").place(x=300, y=280)
        finishLabel = tkinter.Label(
            finishFrame, text="Puzzle Has Been Solved !!", width=33,
background="green", foreground="white")
        finishLabel.place(x=320, y=297)
        timeExecLabel = tkinter.Label(
            finishFrame, text=f"Time Execution: {end-start}", width=33,
background="green", foreground="white")
```

```
        timeExecLabel.place(x=320, y=317)
        nodeRaisedLabel = tkinter.Label(
            finishFrame, text=f"Nodes Raised: {simpulChecked}", width=33,
background="green", foreground="white")
        nodeRaisedLabel.place(x=320, y=337)
        outputFile.write("Puzzle Has Been Solved !!\n")
        outputFile.write(f"Time Execution: {end-start} \n")
        outputFile.write(f"Nodes Raised: {simpulChecked} \n")

        idGoal = simpulChecked
        iter = 1
        listLangkah = findLangkah(listSimpul, idGoal)
        for pzl in listLangkah:
            outputFile.write(
                "Langkah ke-" + str(iter) + "\n")
            iter += 1
            printPuzzle(pzl, outputFile)
            root.update()
            time.sleep(1)
            for i in range(4):
                for j in range(4):
                    if(pzl[i][j] != 16):
                        listPuzzleLabel[i*4+j].config(text=str(pzl[i][j]))
                    else:
                        listPuzzleLabel[i*4+j].config(text="")
    else:
        finishFrame = tkinter.Frame(master=root, width=275,
                                    height=100, background="red").place(x=300,
y=280)
        finishLabel = tkinter.Label(
            finishFrame, text="Puzzle Can't Be Solved :(", font="Arial, 12",
width=28, background="red", foreground="white")
        finishLabel.place(x=313, y=314)
        outputFile.write("Puzzle Can't Be Solved :(")


def randomClicked():
    randomList = random.sample(range(1, 17), 16)
    fillPuzzleFromList(puzzle, randomList)
    for i in range(4):
        for j in range(4):
            if(puzzle[i][j] != 16):
                listPuzzleLabel[i*4+j].config(text=str(puzzle[i][j]))
            else:
                listPuzzleLabel[i*4+j].config(text="")
    root.update()


def importClicked():
```

```
    file = open("../test/" + importFileEntry.get(), "r")
    for i in range(4):
        f = file.readline().split()
        for j in range(4):
            puzzle[i][j] = int(f[j])
    for i in range(4):
        for j in range(4):
            if(puzzle[i][j] != 16):
                listPuzzleLabel[i*4+j].config(text=str(puzzle[i][j]))
            else:
                listPuzzleLabel[i*4+j].config(text="")
    file.close()
    root.update()


root = tkinter.Tk()
root.geometry('600x400')

listPuzzleLabel = []
puzzleFrame = tkinter.Frame(master=root, width=250,
                            height=250, background="grey").place(x=25, y=25)

puzzle = [[i*4+j+1 for j in range(4)] for i in range(4)]
for i in range(4):
    for j in range(4):
        frame = tkinter.Frame(
            root, relief=tkinter.RAISED, borderwidth=2)
        a = (50+50*j)
        b = (50+50*i)
        frame.place(x=a, y=b)
        if(puzzle[i][j] != 16):
            label = tkinter.Label(master=frame, width=6,
                                  height=3, text=str(puzzle[i][j]))
        else:
            label = tkinter.Label(master=frame, width=6,
                                  height=3, text="")
        listPuzzleLabel.append(label)
        label.pack()
root.update()

randomButton = tkinter.Button(
    root, text="Randomize", width=10, command=randomClicked)
randomButton.place(x=25, y=280)

lbl = tkinter.Label(root, text="or")
lbl.place(x=125, y=280)

importFileEntry = tkinter.Entry(root, width=18)
importFileEntry.place(x=163, y=280)
```

```
importButton = tkinter.Button(
    root, text="IMPORT", width=10, command=importClicked)
importButton.place(x=125, y=320)

startButton = tkinter.Button(
    root, text="START", width=10, command=startSolver)
startButton.place(x=25, y=320)

root.mainloop()
```

tc1.txt

1 2 3 4

5 6 16 8

9 10 7 11

13 14 15 12

tc2.txt

5 1 3 4

9 2 7 8

16 6 15 11

13 10 12 14

tc3.txt

5 1 3 4

9 2 7 8

16 6 15 11

13 12 10 14

tc4.txt

2 1 3 4

5 6 16 8

9 10 7 11

13 14 15 12

tc5.txt

1 6 2 4

5 16 3 8

9 7 15 11

13 14 10 12

Link Github: https://github.com/sumertayoga/Tucil-3-Stima.git