Laporan

Tugas Kecil 3 IF2211 Strategi Algoritma

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

Disusun Oleh:

Kent Liusudarso - 13520069

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

ALGORITMA BRANCH AND BOUND

1. Cek apakah puzzle dapat diselesaikan dengan rumus $\sum_{n-1}^{16} KURANG(i) + X$ dimana nilai X adalah 1 jika sel kosong berada pada ubin ganji dan 0 jika sel kosong berada pada ubin genap. Selain itu, fungsi $KURANG(i)$ merupakan banyaknya ubin bernomor j sedemikian sehingga $j < i$ dan $POSISI(j) > POSISI(i)$. Dimana fungsi $POSISI(i)$ merupakan posisi ubin bernomor i pada susunan yang diperiksa.
2. Jika rumus tersebut menghasilkan nilai genap, maka puzzle dapat diselesaikan
3. Selanjutnya, *cost* setiap simpul dihitung dengan $\hat{c}(i) = \hat{f}(i) + \hat{g}(i)$ dimana $\hat{f}(i)$ merupakan kedalaman pohon dan $\hat{g}(i)$ merupakan jumlah ubin yang tidak terdapat pada susunan akhir.
4. Pertama, buat branch dari posisi awal simpul yang menghasilkan simpul baru dari simpul awal yang ubin kosongnya digeser.
5. Jumlah simpul baru yang dihasilkan ditentukan dengan gerakan atas, kanan, bawah, kiri
6. Jika simpul yang ingin dibangkitkan melanggar aturan seperti gerakan ke atas padahal ubin kosongnya sudah dipaling atas atau gerakan ke bawah padahal gerakan sebelumnya keatas (balik lagi) maka, simpul tersebut tidak dibangkitkan.
7. Jika node yang dihasilkan belum sama dengan goal node (15 puzzle yang selesai), maka dilanjutkan dengan menentukan node dengan cost terendah dari node – node yang sudah dibangkitkan untuk dibranch selanjutnya.
8. Jika sudah ditemukan goal node, program selesai.

SOURCE CODE

Bahasa Pemrograman: Python

puzzle.py

```
import random

DIM = 4 # dimension of puzzle
SIZE = DIM * DIM # board size
BLANK = "-" # blank tile

# Goal Node
GOAL = [["1", "2", "3", "4"],
        ["5", "6", "7", "8"],
        ["9", "10", "11", "12"],
        ["13", "14", "15", BLANK]]
PATH = "./test/" # path to puzzle files

# create shuffled 15 puzzle
def createShuffledPuzzle():
    value = ["1", "2", "3", "4",
             "5", "6", "7", "8",
             "9", "10", "11", "12",
             "13", "14", "15", BLANK]
    random.shuffle(value)
    puzzle = []
    it = 0
    for i in range(DIM):
        row = []
        for j in range(DIM):
            row.append(value[it])
            it += 1
        puzzle.append(row)
    return puzzle

# read puzzle from file
def readPuzzle(filename):
    puzzle = []
    with open(filename, "r") as f:
        for line in f:
            row = []
            for i in line.split():
                row.append(i)
            puzzle.append(row)
    return puzzle

# print puzzle
def displayPuzzle(puzzle):
    for i in range(DIM):
```

```
        for j in range(DIM):
            print(puzzle[i][j], end=" ")
        print()
```

solver.py

```
from platform import node
from puzzle import *
import copy

def POSISI(num, puzzle):
    if(num == 16):
        i, j = findBlank(puzzle)
        return i * DIM + j
    for i in range(DIM):
        for j in range(DIM):
            if puzzle[i][j] == str(num):
                return i * DIM + j

def KURANG(num, puzzle):
    count = 0
    for i in range(1, SIZE + 1):
        if POSISI(num, puzzle) < POSISI(i, puzzle):
            if(num > i):
                count += 1
    return count

def isSolvable(puzzle):
    row, col = findBlank(puzzle)
    if (row + col) % 2 == 0:
        X = 0
    else:
        X = 1

    totalKURANG = 0
    for i in range (1, SIZE + 1):
        totalKURANG += KURANG(i, puzzle)

    if (totalKURANG + X) % 2 == 0:
        return True, totalKURANG + X
    else:
        return False, totalKURANG + X

def isSolved(puzzle):
    return puzzle == GOAL

def swap(puzzle, row1, col1, row2, col2):
```

```
    puzzle[row1][col1], puzzle[row2][col2] = puzzle[row2][col2],
puzzle[row1][col1]


def findBlank(puzzle):
    for i in range(DIM):
        for j in range(DIM):
            if puzzle[i][j] == BLANK:
                return i, j

def moveBlank(puzzle, move):
    tempPuzzle = copy.deepcopy(puzzle)
    row, col = findBlank(tempPuzzle)
    if move == "up":
        swap(tempPuzzle, row, col, row - 1, col)
    elif move == "down":
        swap(tempPuzzle, row, col, row + 1, col)
    elif move == "left":
        swap(tempPuzzle, row, col, row, col - 1)
    elif move == "right":
        swap(tempPuzzle, row, col, row, col + 1)
    return tempPuzzle

def availableMove(puzzle, lastMove):
    availableMoves = []
    row, col = findBlank(puzzle)
    if row > 0:
        availableMoves.append('up')
    if row < DIM - 1:
        availableMoves.append('down')
    if col > 0:
        availableMoves.append('left')
    if col < DIM - 1:
        availableMoves.append('right')

    delOppositeMove(availableMoves, lastMove)
    return availableMoves

def delOppositeMove(moves, lastMove):
    if(lastMove == "up"):
        moves.remove("down")
    elif(lastMove == "down"):
        moves.remove("up")
    elif(lastMove == "left"):
        moves.remove("right")
    elif(lastMove == "right"):
        moves.remove("left")

def getCost(puzzle, depth):
```

```
    cost = 0
    for i in range(DIM):
        for j in range(DIM):
            if puzzle[i][j] != GOAL[i][j] and puzzle[i][j] != BLANK:
                cost += 1
    return cost + depth

class Node:
    def __init__(self, data=None):
        self.puzzle = data
        self.parent = None
        self.depth = 0

class PrioQueue(object):
    def __init__(self):
        self.queue = []

    def __str__(self):
        return '\n'.join([str(i) for i in self.queue])

    def enqueue(self, data):
        self.queue.append(data)

    def dequeue(self):
        temp = 0
        for i in range(len(self.queue)):
            if(self.queue[i][0] < self.queue[temp][0]):
                temp = i
        item = self.queue[temp]
        del self.queue[temp]
        return item

def displayPath(node):
    if(node.parent == None):
        return
    displayPath(node.parent)
    print("Move ",node.depth,": ")
    displayPuzzle(node.puzzle)
    print()

def solver(puzzle, totalNode):
    node = Node(puzzle)
    queue = PrioQueue()
    # [cost, node, move]
    queue.enqueue([getCost(node.puzzle, node.depth), node, ""])
    currentNode = Node(puzzle)
    while not isSolved(currentNode.puzzle):
        minCostNode = queue.dequeue()
```

```
        currentNode = minCostNode[1]
        lastMove = minCostNode[2]

        if isSolved(currentNode.puzzle):
            return currentNode, totalNode

        availableMoves = availableMove(currentNode.puzzle, lastMove)

        # Generate new node
        newDepth = currentNode.depth + 1
        for move in availableMoves:
            totalNode += 1
            newNode = Node(moveBlank(currentNode.puzzle, move))
            newNode.parent = currentNode
            newNode.depth = newDepth
            newCost = getCost(newNode.puzzle, newDepth)

            queue.enqueue([newCost, newNode, move])
```

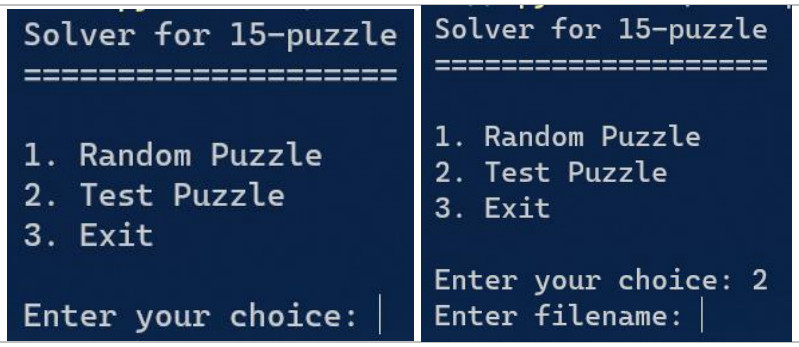
main.py

```
import os
import time
from solver import *

print("Solver for 15-puzzle")
print("====================")
print()
print("1. Random Puzzle")
print("2. Test Puzzle")
print("3. Exit")
print()
choice = 0
while(choice != 1 and choice != 2 and choice != 3):
    choice = int(input("Enter your choice: "))
    if choice == 1:
        puzzle = createShuffledPuzzle()
    elif choice == 2:
        filename = input("Enter filename: ")
        puzzle = readPuzzle(PATH + filename)
    elif choice == 3:
        exit()
    else:
        print("Invalid choice")

os.system("cls")
print("Puzzle:")
```

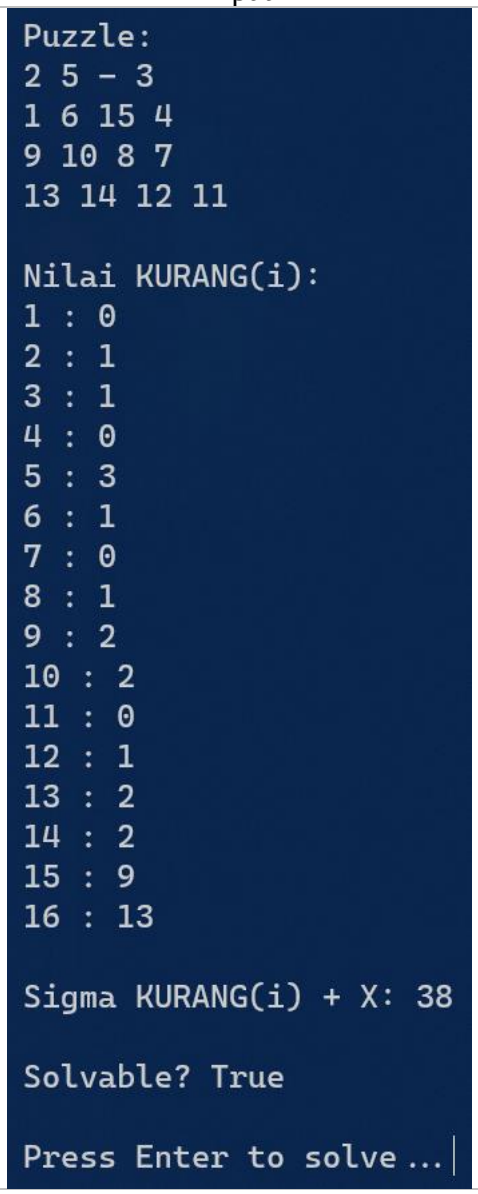
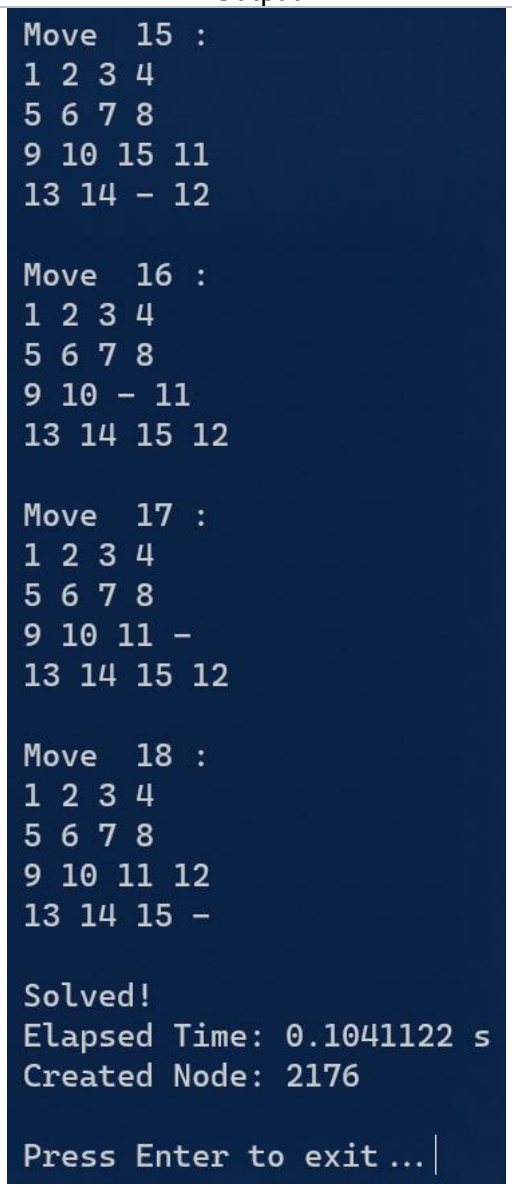
```
displayPuzzle(puzzle)
print()
print("Nilai KURANG(i):")
for i in range(1, SIZE + 1):
    # if KURANG(i, puzzle) != 0:
        print(i, ":", KURANG(i, puzzle))
isSolve, totalKURANG = isSolvable(puzzle)
print()
print("Sigma KURANG(i) + X:", totalKURANG)
print()
print("Solvable?", isSolve)
print()

if(isSolve):
    input("Press Enter to solve...")
    os.system("cls")
    print("Solving...")
    print()
    timerStart = time.perf_counter()
    pathNode, totalNode = solver(puzzle, totalNode = 0)
    timerEnd = time.perf_counter()
    displayPuzzle(puzzle)
    displayPath(pathNode)
    print("Solved!")
    print(f"Elapsed Time: {timerEnd - timerStart:0.7f} s")
    print("Created Node:", totalNode)
    print()
    input("Press Enter to exit...")
else:
    input("Press Enter to exit...")
```

SCREENSHOT INPUT OUTPUT

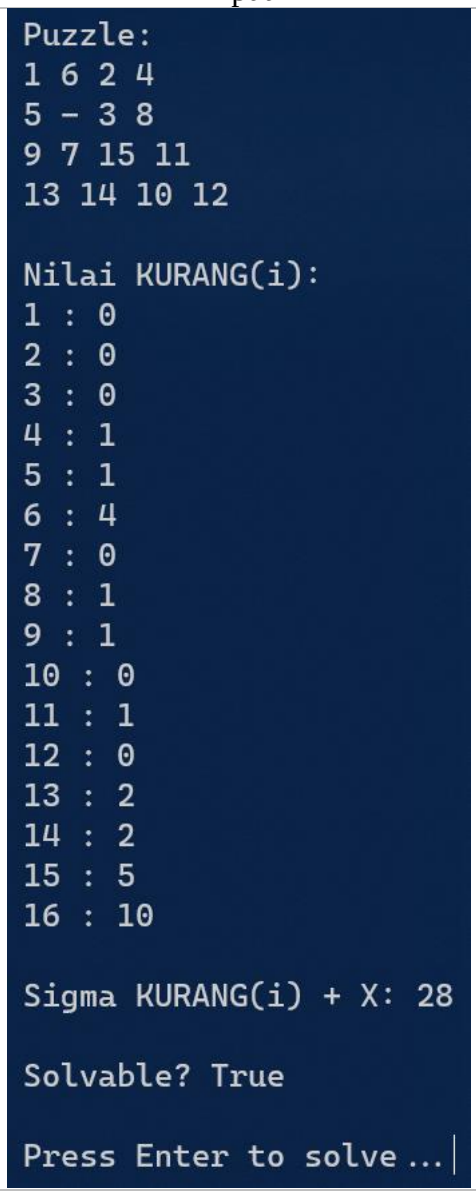
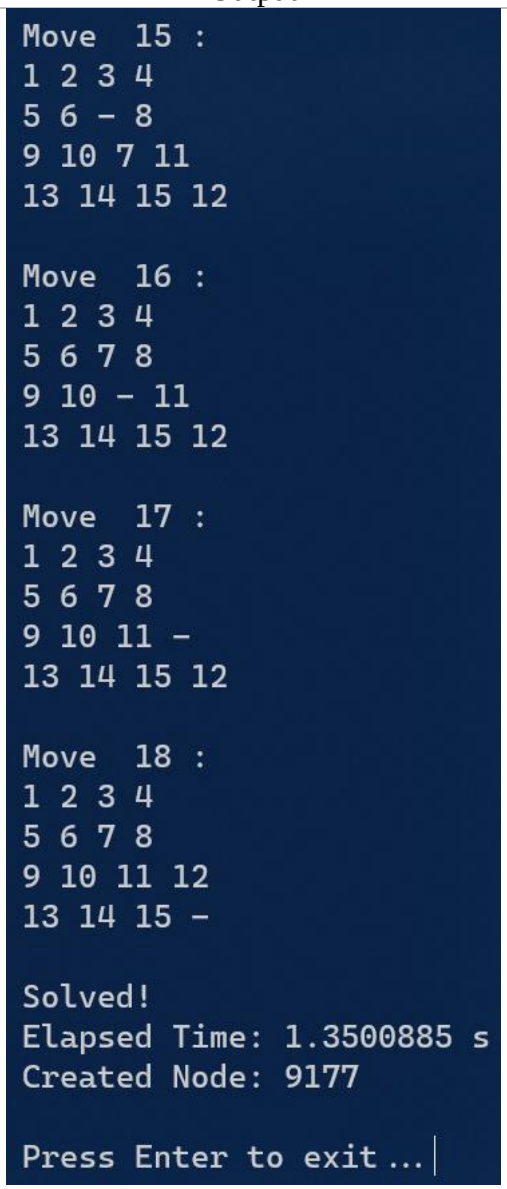
1. Main Menu

Main


2. solvable1.txt

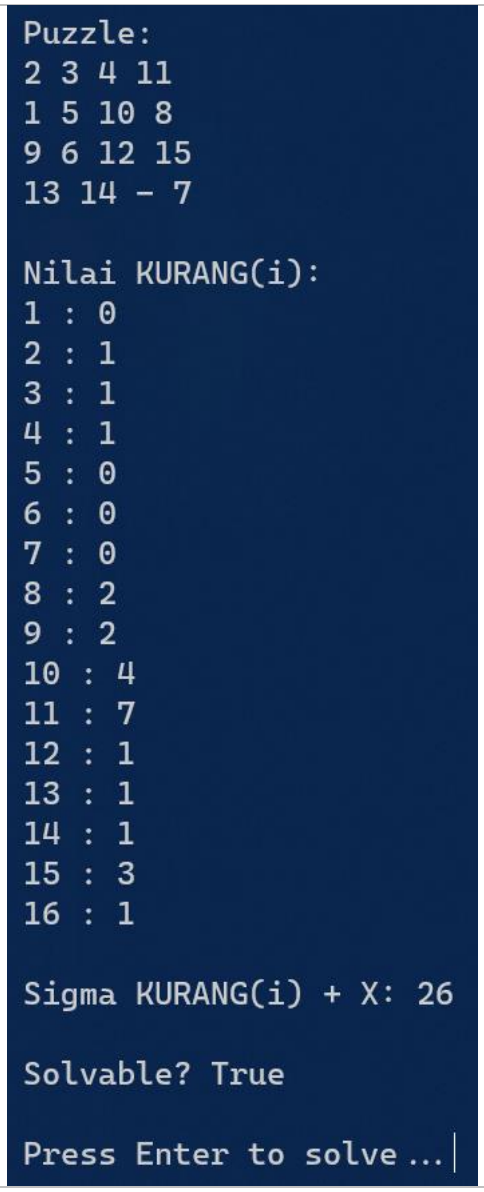
Input	Output
	

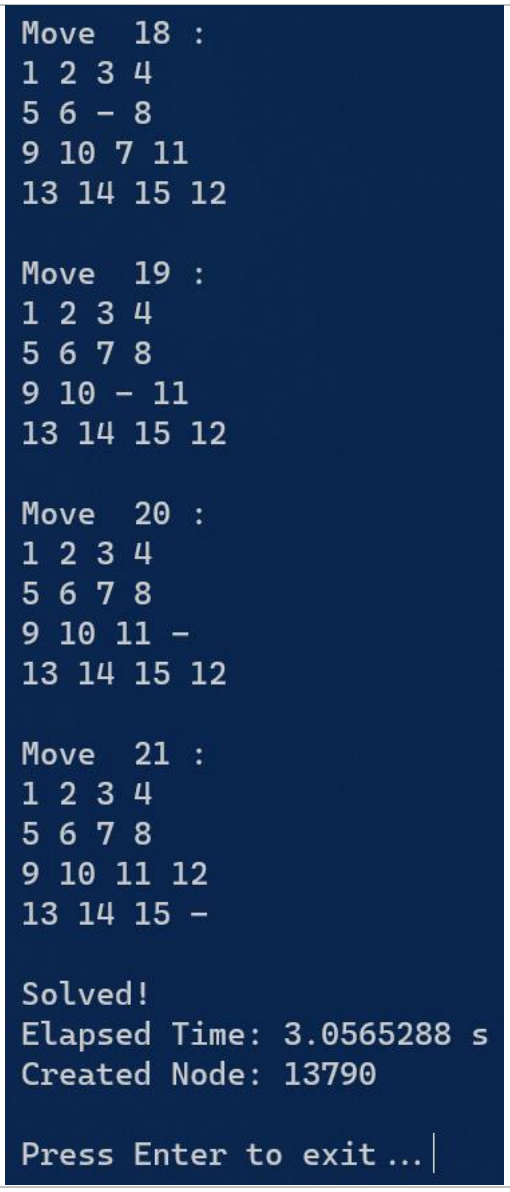
3. solvable2.txt

Input	Output
	

4. solvable3.txt

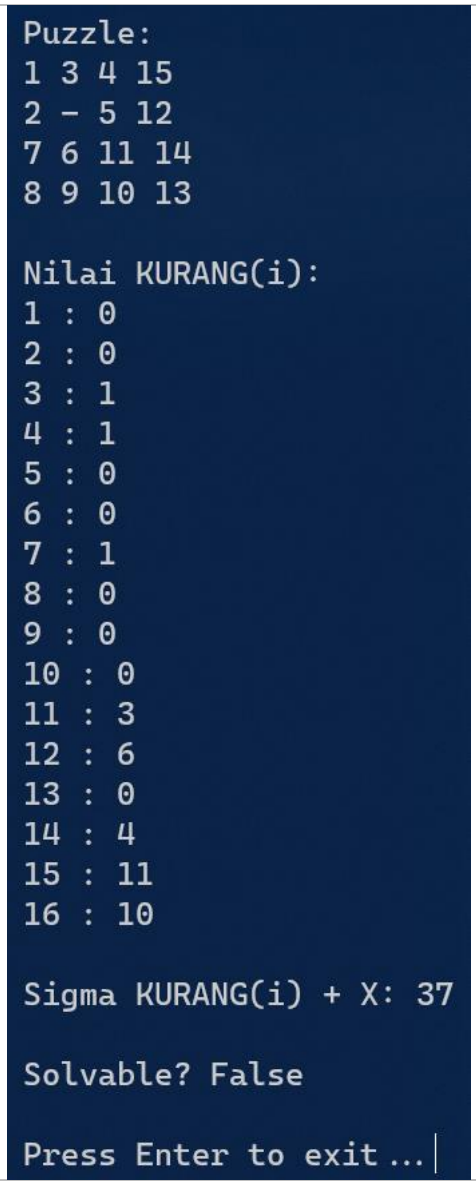
Input	Output





5. unsolvable1.txt

Input	Output

6. unsolvable2.txt

Input	Output

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

Nilai KURANG(i):
1 : 0
2 : 0
3 : 2
4 : 0
5 : 3
6 : 4
7 : 2
8 : 2
9 : 8
10 : 4
11 : 6
12 : 5
13 : 7
14 : 1
15 : 0
16 : 6

Sigma KURANG(i) + X: 51

Solvable? False

Press Enter to exit ...
```

SOURCE CODE FILE

https://github.com/kentlius/Tucil3_13520069

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		√