Laporan Tugas Kecil 1

IF2211 Strategi Algoritma

Penyelesaian Permainan Queens Linkedin

Semester II Tahun 2025/2026

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan InformatikaPendahuluan

1. Pendahuluan

Queens adalah permainan pada situs web LinkedIn yang dirilis setiap hari. Pada permainan ini, pemain diminta untuk meletakkan sebuah mahkota ratu (Queen) pada setiap masing-masing daerah yang ditandai dengan warna berbeda. Tidak boleh ada dua atau lebih mahkota yang diletakkan di baris, kolom, atau warna yang sama. Selain itu, sekeliling mahkota juga harus kosong, secara vertikal, horizontal, maupun diagonal.

Pada tugas kecil ini, saya akan mengimplementasikan algoritma yang akan mencari solusi untuk permainan *Queens* dengan metode *brute force*. Implementasi dilakukan menggunakan bahasa Python dengan mempertimbangkan kecepatan dan kesederhanaan kode.

Melalui tugas besar ini, saya berharap dapat memahami algoritma *brute force* secara mendalam dan cara mengimplementasikannya secara nyata melalui permainan *Queens*. Saya juga berharap agar pengguna dapat menggunakan program saya untuk menyelesaikan permainan *Queens* dengan menampilkan solusi yang valid dan berhasil divisualisasikan agar lebih mudah dipahami.

2. Implementasi

readWriteFile.py

```
import os

def readFile(fileName):
    if os.path.exists(fileName):
        with open(fileName, "r") as f:
            return f.readlines()

    print("File tidak ada!")
    return

def writeFile(fileName, listData):
    f = open(fileName, "w")

    for row in listData:
        f.write("".join(row) + "\n")

    f.close()
```

dataManipulation.py

```
def cleanData(listData):
    for i in range(len(listData)):
        listData[i] = listData[i].strip()
        charList = [char for char in listData[i]]
        listData[i] = charList

    return listData

def printData(listData):
    for row in listData:
        print("".join(row))

def isDataValid(listData):
    if len(listData) == 0:
        return False
    for i in range (len(listData)):
        if len(listData[i]) != len(listData):
            return False
    for i in range(0, len(listData)):
        for j in range(0, len(listData)):
            if listData[i][j] == " ":
                return False
    return True
```

solveQueen.py

```
def permutations(arr, r):
    arr = list(arr)
    if r == 0:
        yield []
    else:
        for i in range(len(arr)):
            rest = arr[:i] + arr[i+1:]
            for p in permutations(rest, r-1):
                yield [arr[i]] + p

# Hitung jumlah Queens
def numberOfColors(listData):
    l1 = []
    count = 0
    for i in range (len(listData)):
        for char in listData[i]:
            if char not in l1:
                count +=1
                l1.append(char)
    return count

# Generate Position
```

```
def generateSquarePosition(count):
    l = []
    a = 0
    for i in range (count):
        for j in range (count):
            l.append(a)
            a += 1

    return l


def valid_warna(square, ori):
    warnaTerpakai = set()
    n = len(square)

    for i in range(n):
        for j in range(n):
            if square[i][j] == '#':
                warna = ori[i][j]
                if warna in warnaTerpakai:
                    return False
                warnaTerpakai.add(warna)

    return True

def pemeriksaan(listData):
    #pemeriksaan horizontal
    for i in range (len(listData)):
        horizontalTag = False
        for j in range (len(listData[i])):
            if (listData[i][j] == '#') and (horizontalTag != True):
                horizontalTag = True
            elif (listData[i][j] == '#') and (horizontalTag == True):
                return False

            if (listData [i][j] == '#'):
                if (i-1 >= 0 and j-1>=0):
                    if listData[i-1][j-1] == '#':
                        return False

                if (i-1 >= 0 and j+1 < len(listData)):
                    if (listData[i-1][j+1] == '#'):
                        return False

                if (i+1 < len(listData) and j-1 >= 0):
                    if (listData[i+1][j-1] == '#'):
                        return False

                if (i+1 < len(listData) and j+1 < len(listData)):
                    if (listData[i+1][j+1] == '#'):
                        return False

    #pemeriksaan vertikal
    for j in range (len(listData)):
        verticalTag = False
        for i in range (len(listData[i])):
            if (listData[i][j] == '#') and (verticalTag != True):
                verticalTag = True
            elif (listData[i][j] == '#') and (verticalTag == True):
                return False

    return True

def solveMurni(listData, visualCallback=None):
    jumQueen = numberOfColors(listData)
    lPosition = generateSquarePosition(len(listData))
    iteration = 0
    for p in permutations(lPosition, jumQueen):
        iteration += 1
        square = [list(row) for row in listData]

        for i in range(len(p)):
            row = p[i]//len(listData)
            column = p[i] % len(listData)
```

```
            square[row][column] = '#'
        if visualCallback and iteration % 100 == 0:
            visualCallback(square)

        if pemeriksaan(square) and valid_warna(square, listData):
            return square, iteration

    return None, iteration


def solve(listData, visualCallback=None):
    iteration = 0
    n = len(listData)
    r = numberOfColors(listData)
    for cols in permutations(range(n), r):
        iteration += 1
        square = [list(row) for row in listData]

        for row in range(r):
            square[row][cols[row]] = '#'
        if visualCallback and iteration % 100 == 0:
            visualCallback(square)
        if pemeriksaan(square) and valid_warna(square, listData):
            return square, iteration
    return None, iteration
```

appView.py

```
import tkinter as tk
import random
import colorsys
import time
from tkinter import filedialog
from PIL import Image, ImageDraw, ImageFont
from src.logic.readWriteFile import readFile, writeFile
from src.logic.dataManipulation import cleanData, printData, isDataValid
from src.logic.solveQueen import solve, solveMurni

cleaned = None
currentQueens = None
currentResult = None
isSolving = False

def updateVisual(tempBoard):
    queens = []
    n = len(tempBoard)

    for i in range(n):
        for j in range(n):
            if tempBoard[i][j] == '#':
                queens.append((i, j))

    displayBoard(cleaned, queens)
    root.update()

def saveAsImage():
    global cleaned, mapColor, currentQueens

    if not cleaned:
        statusLabel.config(text="Board belum di-input.", fg="red")
        return

    n = len(cleaned)
```

```
    cell_size = 80

    img = Image.new("RGB", (n*cell_size, n*cell_size), "white")
    draw = ImageDraw.Draw(img)

    font = ImageFont.truetype("seguisym.ttf", int(cell_size/2))

    for i in range(n):
        for j in range(n):
            char = cleaned[i][j]
            color = mapColor[char]

            x0 = j * cell_size
            y0 = i * cell_size
            x1 = x0 + cell_size
            y1 = y0 + cell_size

            draw.rectangle([x0, y0, x1, y1], fill=color, outline="black")

            if currentQueens and (i, j) in currentQueens:
                bbox = draw.textbbox((0, 0), "♛", font = font)
                w = bbox[2] - bbox[0]
                h = bbox[3] - bbox[1]
                draw.text(
                    (x0 + (cell_size - w)/2, y0 + (cell_size - h)/2),
                    "♛",
                    fill="black",
                    font=font
                )

    filePath = filedialog.asksaveasfilename(defaultextension=".jpeg",
filetypes=[("JPEG files", "*.jpeg"), ("All files", "*.*")])
    if filePath:
        img.save(filePath, "JPEG")
        statusLabel.config(text=f"Board berhasil disimpan sebagai {filePath}",
fg="green")

def randomColor():
    h = random.random()
    s = random.uniform(0.2, 0.5)
    v = 0.7 + 0.3*random.random()

    r,g,b = colorsys.hsv_to_rgb(h, s, v)
    return f"#{int(r*255):02x}{int(g*255):02x}{int(b*255):02x}"

def displayBoard(board, queens = None):
    global currentQueens, boardFrame
    currentQueens = queens

    boardFrame.destroy()
    boardFrame = tk.Frame(root)
    boardFrame.pack(before=caraSolveFrame)
    for widget in boardFrame.winfo_children():
        widget.destroy()

    n = len(board)
    global mapColor
    cells = [[None for _ in range(n)] for _ in range(n)]

    for i in range(n):
        boardFrame.grid_rowconfigure(i, weight=1, uniform="row")
        boardFrame.grid_columnconfigure(i, weight=1, uniform="col")
        for j in range(n):
            char = board[i][j]
            color = mapColor[char]
            if queens and (i,j) in queens:
                label = tk.Label(boardFrame, font=("Segoe UI Symbol", 14), text="♛",
borderwidth=1, relief="solid", bg=color)
            else:
                label = tk.Label(boardFrame, font=("Segoe UI Symbol", 14), text =
"     ", borderwidth=1, relief="solid", bg=color)
            label.grid(row=i, column=j, sticky="nsew")
            cells[i][j] = label
```

```
def onInputButton():
    if isSolving:
        statusLabel.config(text="ERROR: Proses masih berlangsung.", fg="red")
        return
    fileName = filedialog.askopenfilename(defaultextension=".txt", filetypes=[("Text
files", "*.txt")])
    lines = readFile(fileName)
    if lines:
        statusLabel.config(text="File ditemukan!", fg="green")
        global cleaned, mapColor
        cleaned = cleanData(lines)
        validity = isDataValid(cleaned)

        if (validity):
            printData(lines)
            print("")
            chars = set(char for row in cleaned for char in row)
            mapColor = {char: randomColor() for char in chars}
            displayBoard(cleaned)
            timeLabel.config(text= "")
            iterationLabel.config(text= "")
        else:
            print("Data tidak valid!")
            statusLabel.config(text="Data tidak valid!", fg="red")

    else:
        statusLabel.config(text="File tidak ada!", fg="red")

def runSolve1():
    global cleaned, currentResult, isSolving
    if not cleaned:
        statusLabel.config(text="Board belum di-input.", fg="red")
        return
    if isSolving:
        statusLabel.config(text="ERROR: Proses masih berlangsung.", fg="red")
        return
    isSolving = True
    statusLabel.config(text="Sedang mencari jawaban...", fg="orange")
    timeLabel.config(text=f"Waktu pencarian: Loading...")
    iterationLabel.config(text=f"Banyak kasus yang ditinjau: Loading...")
    root.update_idletasks()
    startTime = time.time()
    result, iteration = solve(cleaned, updateVisual)

    if result:
        currentResult = result
        queens = [(i, j) for i in range(len(result)) for j in range(len(result)) if
result[i][j] == '#']
        print("=======================")
        print("Hasil:")
        print("")
        displayBoard(cleaned, queens)
        printData(result)
        print("")
        statusLabel.config(text="Jawaban ditemukan!", fg="green")
        root.update()

    else:
        statusLabel.config(text="Tidak ada jawaban.", fg="red")
        displayBoard(cleaned)
    endTime = time.time()
    totalTime = (endTime - startTime) * 1000
    print(f"Banyak kasus yang ditinjau: {iteration} kasus")
    print(f"Waktu pencarian: {totalTime:.2f} ms")
    timeLabel.config(text=f"Waktu pencarian: {totalTime:.2f} ms")
    iterationLabel.config(text=f"Banyak kasus yang ditinjau: {iteration} kasus")
    isSolving = False

def runSolve2():
    global cleaned, currentResult, isSolving
    if not cleaned:
        statusLabel.config(text="Board belum di-input.", fg="red")
```

```
        return
    if isSolving:
        statusLabel.config(text="ERROR: Proses masih berlangsung.", fg="red")
        return
    isSolving = True
    statusLabel.config(text="Sedang mencari jawaban...", fg="orange")
    timeLabel.config(text=f"Waktu pencarian: Loading...")
    iterationLabel.config(text=f"Banyak kasus yang ditinjau: Loading...")
    root.update_idletasks()
    startTime = time.time()
    result, iteration = solveMurni(cleaned, updateVisual)

    if result:
        currentResult = result
        queens = [(i, j) for i in range(len(result)) for j in range(len(result)) if
result[i][j] == '#']
        displayBoard(cleaned, queens)
        print("======================")
        print("Hasil:")
        print("")
        printData(result)
        print("")
        statusLabel.config(text=f"Jawaban ditemukan!", fg="green")
        root.update()

    else:
        statusLabel.config(text="Tidak ada jawaban.", fg="red")
        displayBoard(cleaned)
    endTime = time.time()
    totalTime = (endTime - startTime) * 1000
    print(f"Banyak kasus yang ditinjau: {iteration} kasus")
    print(f"Waktu pencarian: {totalTime:.2f} ms")
    timeLabel.config(text=f"Waktu pencarian: {totalTime:.2f} ms")
    iterationLabel.config(text=f"Banyak kasus yang ditinjau: {iteration} kasus")
    isSolving = False

def saveToFile():
    global currentResult

    if not currentResult:
        statusLabel.config(text="Board belum di-input.", fg="red")
        return

    filePath = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text
files", "*.txt"), ("All files", "*.*")])
    if filePath:
        writeFile(filePath, currentResult)
        statusLabel.config(text=f"Board berhasil disimpan sebagai {filePath}",
fg="green")

def GUI():
    global root, boardFrame, statusLabel, timeLabel, iterationLabel, caraSolveFrame,
inputEntry
    root = tk.Tk()

    root.title("Queens Game Solver")
    root.geometry("800x500")

    title = tk.Label(root, text ="Queens Game Solver", font=("Arial", 18))
    title.pack(pady=30)

    inputButton = tk.Button(root, text="Open TXT File", font=("Arial", 14),
command=onInputButton)
    inputButton.pack(pady="5px")

    statusLabel = tk.Label(root, text="", font=("Arial", 8))
    statusLabel.pack()

    boardFrame = tk.Frame(root)
    boardFrame.pack()

    #Pilih cara Solve
    caraSolveFrame = tk.Frame(root)
    caraSolveFrame.pack(pady="5px")
```

```
    solve1Button = tk.Button(caraSolveFrame, text="Algoritma 1", font = ("Arial", 14),
command = runSolve1)
    solve1Button.pack(side = tk.LEFT)

    solve2Button = tk.Button(caraSolveFrame, text="Algoritma 2", font = ("Arial", 14),
command = runSolve2)
    solve2Button.pack(padx="5px")

    penjelasanLabel = tk.Label(root, text = "(Algoritma 1 lebih cepat dari 2)", font =
("Arial", 8))
    penjelasanLabel.pack()

    penjelasan2Label = tk.Label(root, text = "(Untuk  menghentikan pencarian, quit
software)", font = ("Arial", 8))
    penjelasan2Label.pack()

    saveFrame = tk.Frame(root)
    saveFrame.pack(pady=5)

    inputButton = tk.Button(saveFrame, text="Save to TXT File", font=("Arial", 14),
command=saveToFile)
    inputButton.pack(side = tk.LEFT)

    downloadImageButton = tk.Button(saveFrame, text="Download as JPEG", font=("Arial",
14), command=saveAsImage)
    downloadImageButton.pack(padx = "5px")

    timeLabel = tk.Label(root, text="", font= ("Arial",14))
    timeLabel.pack(pady = "5px")
    iterationLabel = tk.Label(root, text="", font=("Arial", 14))
    iterationLabel.pack()

    root.mainloop()
```

3. Penjelasan Algoritma *Brute Force*

1. Algoritma 1: Brute Force Berbasis Kolom

```
function solve(listData: list of list of string, visualCallback: boolean) → list
of list of string, integer
{ Mencari solusi dari permainan queens dengan melakukan brute force berbasis
kolom }

KAMUS LOKAL
        function numberOfColors (listData: list of list of string) → integer
        { Menghitung jumlah warna yang ada }

        function permutations(arr: list of integer, r: integer) → list of list of
        integer
        { Menghasilkan semua permutasi panjang r dari arr }

        function pemeriksaan(listData: list of list of string) → boolean
        { Memeriksa apakah papan memenuhi aturan posisi }

        function valid_warna(square: list of list of string, ori: list of list of
        string) → boolean
        { Memeriksa apakah papan memenuhi aturan warna }


        n, r, i, j, iteration: integer
        listPerm : list of list of integer
        p : list of integer
        square : list of list of string

ALGORITMA

        visualCallback ← None
        iteration ← 0
        n ← len(listData)
        r ← numberOfColors(listData)

        listPerm ← permutations(range(n), r)

        p traversal[0..len(listPerm)-1]
                iteration ← iteration + 1

                i traversal[0..n-1]
                        j traversal[0..n-1]
                                square[i][j] ← listData[i][j]

                i traversal[0..r-1]
                        square[i][p[i]] ← '#'

                if (visualCallback) and (iteration mod 100 = 0) then
                        visualCallback(square)

                if (pemeriksaan(square)) and (valid_warna(square, listData)) then
                        → square, iteration

        → None, iteration
```

Fungsi **solve** bertujuan untuk mencari solusi dari permainan *Queens* dengan metode *brute force* berbasis kolom. Fungsi ini lebih cepat dan efisien dari metode *brute force* naif karena ruang pencarian dikurangi. Fungsi ini menerima dua parameter, yaitu **listData** yang merupakan papan permainan dan **visualCallback** yang digunakan untuk untuk menampilkan proses visualisasi. Kemudian, fungsi ini akan mengembalikan **square** yang merupakan solusi dari permainan *Queens* dalam

bentuk papan dan **iteration** yang merupakan jumlah berapa banyak kasus yang ditelusuri.

Pertama, variabel **iteration** diinisialisasi dengan nilai 0. Variabel **n** diisi dengan panjang dari **listData**, sementara variabel **r** diisi dengan jumlah berapa banyak warna yang unik dalam papan. Kemudian, permutasi dilakukan untuk menentukan semua kemungkinan posisi mahkota, dengan setiap permutasi merepresentasikan kemungkinan posisi kolom untuk setiap baris. Pada setiap iterasi, papan square dibuat sebagai salinan dari papan asli yang bersifat sementara. Setelah itu, mahkota diletakkan ke dalam papan. Untuk setiap 100 iterasi, papan sementara akan dikirimkan ke fungsi visualCallback untuk keperluan visualisasi pada GUI. Papan sementara lalu dicek apakah memenuhi persyaratan. Jika papan memenuhi persyaratan, makan fungsi akan mengembalikan papan tersebut dan jumlah iterasi yang telah dilakukan. Jika tidak, iterasi akan berlanjut ke permutasi berikutnya. Apabila semua kemungkinan sudah diperiksa dan masih belum menemukan solusi, fungsi akan mengembalikan nilai **None** dan jumlah iterasi terakhir. Hal ini berarti persoalan tidak memiliki solusi.

2. Algoritma 2: Brute Force Naif

```
function solveMurni (listData: list of list of string, visualCallback : boolean)
→ list of list of string, integer
{ Melakukan brute force secara naif }

KAMUS LOKAL
        function numberOfColors (listData: list of list of string) → integer
        { Menghitung jumlah warna yang ada }

        function generateSquarePosition(count: integer) → list of integer
        { Menghasilkan list posisi pada board }

        function permutations(arr: list of integer, r: integer) → list of list of
        integer
        { Menghasilkan semua permutasi panjang r dari arr }

        function pemeriksaan(listData: list of list of string) → boolean
        { Memeriksa apakah papan memenuhi aturan posisi }

        function valid_warna(square: list of list of string, ori: list of list of
        string) → boolean
        { Memeriksa apakah papan memenuhi aturan warna }

        jumQueen, iteration, i, j : integer
        lPosition, p : list of integer
        listPerm : list of list of integer
        square : list of list of string

ALGORITMA

        visualCallback ← None
        jumQueen ← numberOfColors(listData)
        lPosition ← generateSquarePosition(len(listData))
        iteration ←  0

        listPerm ← permutations(lPosition, jumQueen)
        p traversal[0..len(listPerm)-1]
                iteration ← iteration + 1

                i traversal[0..getRowEff(listData)-1]
                        j traversal[0..getColEff(listData[i])-1]
                                square[i][j] ← listData[i][j]

                i traversal[0..len(p)-1]
                        row ← p[i] div len(listData)
                        column ← p[i] mod len(listData)
                        square[row][column] ← '#'

                if (visualCallback) and (iteration%100 = 0) then
                        visualCallback(square)

                if (pemeriksaan(square)) and (valid_warna(square,listData)) then
                        → square, iteration

        → None, iteration
```

Fungsi solveMurni adalah *brute force* naif, yaitu memeriksa semua kemungkinan yang ada tanpa membatasi baris dan kolom sejak awal. Sama seperti fungsi sebelumnya, fungsi ini menerima dua parameter, yaitu **listData** dan **visualCallback** Fungsi ini juga akan mengembalikan **square** yang merupakan solusi dari permainan

Queens dalam bentuk papan dan **iteration** yang merupakan jumlah berapa banyak kasus yang ditelusuri.

Pada awal agoritma, jumlah mahkota yang harus ditempatkan dihitung dan daftar indeks posisi linear dari seluruh sel pada papan dibuat. Variabel **iteration** juga diinisialisasikan dengan nilai 0. Setiap permutasi **p** merepresentasikan kombinasi posisi queen yang akan dicoba. **Square** adalah papan sementara yang merupakan salinan dari papan asli. Mahkota akan diletakkan pada papan sementara dengan menggunakan operasi pembagian dan modulo. Untuk setiap 100 iterasi, **visualCallback** akan mengirimkan papan sementara untuk divisualisasikan. Papan yang sudah diperiksa dan memenuhi aturan akan dikembalikan bersama dengan jumlah iterasi yang telah dilakukan. Jika soal yang dimasukkan pengguna tidak memiliki solusi, fungsi akan mengembalikan nilai **None** dan jumlah iterasi yang telah dilakukan.

4. Uji Coba *Test Case*

1.

Input Soal	AABBC ABBCC ABBCC DDEEE DDDEE 
Output Algoritma 1	

Output Algoritma 2	

2.

Input Soal	AAAAA BBBBB CCCCC DDDDD EEEEE 
Output Algoritma 1	
Output Algoritma 2	

3.

Input Soal	ABCDE ABCDE ABCDE ABCDE ABCDE 
Output Algoritma 1	
Output Algoritma 2	

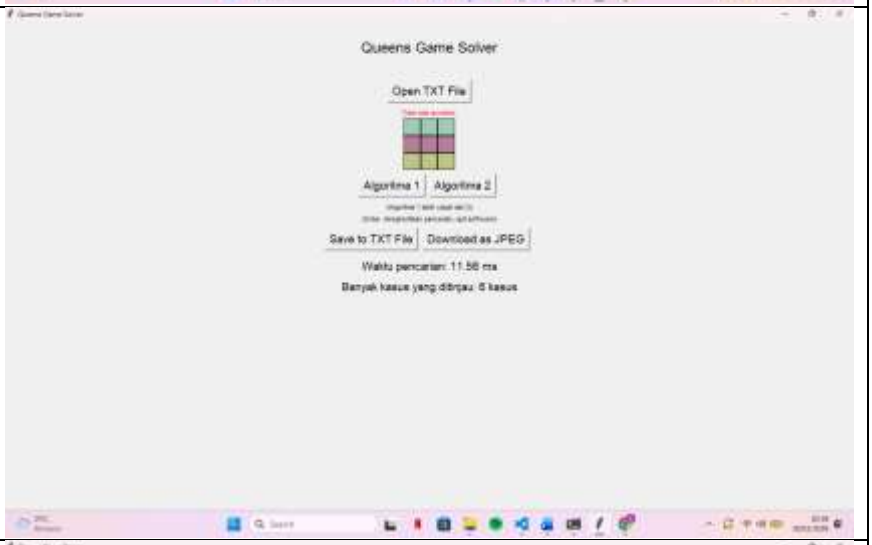
4.

Input Soal	AABBB AACBB ACCCD AECDD EEEED 
Output Algoritma 1	
Output Algoritma 2	

5.

Input Soal	ABACC AAADC AAADC EAADC EEDDC 
Output Algoritma 1	
Output Algoritma 2	

6.

Input Soal	AAA BBB CCC 
Output Algoritma 1	
Output Algoritma 2	

7.

Input Soal	AABBC ABBCC AB CC DDEEEE 

5. Kesimpulan

Program yang telah dibuat memiliki dua jenis algoritma untuk menyelesaikan permainan *Queens* LinkedIn, yaitu algoritma *brute force* berbasis kolom dan *brute force* naif. Kedua algoritma berhasil berjalan dengan baik. Algoritma *brute force* naif memakan waktu proses yang lebih lama dibandingkan *brute force* berbasis kolom.

Pernyataan

Tugas ini disusun sepenuhnya tanpa bantuan kecerdasan buatan (Generative AI), melainkan hasil pemikiran dan analisis mandiri.

Eliana Natalie Widjojo

Lampiran

1. Tabel pengecekan:

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	✓	
2	Program berhasil di jalankan	✓	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	✓	
5	Program memiliki Graphical User Interface (GUI)	✓	
6	Program dapat menyimpan solusi dalam bentuk file gambar	✓	

2. Pranala *repository*: https://github.com/EliWidjojo/Stima_Tucil1_13524116