LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA

Penyelesaian Permainan *N-Queens*
dengan Algoritma *Brute Force* Murni

Disusun oleh:

Kurt Mikhael Purba - 13524065

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2026

DAFTAR ISI

BAB I
DESKRIPSI MASALAH DAN ALGORITMA

1. 1 N-Queens

N-Queens merupakan salah satu permainan dari aplikasi Linkedln. Cara memenangkan permainan ini adalah meletakan setiap ratu pada setiap baris dengan beberapa Batasan, yaitu setiap ratu tidak boleh ditempatkan pada baris yang sama dan tiap ratu tidak boleh berada di +1 area sekitar ratu lainnya. Selain itu, setiap warna harus di isi tepat satu ratu saja. Terdapat banyak metode solusi yang dapat digunakan untuk menyelesaikan permainan ini. Laporan ini akan menjelaskan metode penyelesaian masalah ini dengna metode *Brute Force* murni dan *Brute Force* yang telah teroptimasi.

1.2 Langkah-langkah Penyelesaian Permainan *N-Queens*

Dalam menyelesaikan permainan *N-Queens* berdasarkan algoritma, dapat menggunakan metode *Brute Force* Murni dan *Brute Force* Teroptimasi.

1.2.1 Brute Force Murni

Dalam menyelesaikan permainan ini dengan *Brute Force* murni, kita akan melakukan validasi solusi untuk setiap kombinasi yang ada. Dalam pencarian kombinasi ini, program masih memperbolehkan kasus beberapa ratu berada pada baris atau kolom yang sama. Semua kombinasi ini akan diuji kebenarannya sesuai dengan aturan dasar dari permainan ini. Permasalahan yang ditemukan dalam solusi ini adalah bagaimana kita menyimpan setiap kombinasi. Laporan ini menyimpan setiap kombinasi dalam bentuk *list of tuple* , setiap tuple mewakili koordinat baris dan kolom suatu ratu.

Pseudocode :

Func kombinasiBaris(Lisy of Tuple , integer) → List of list of Tuple {Fungsi yang mereturn seluruh kombinasi koordinat untuk setiap ratu}
func kombinasiBaris(nTuple, length): {Base Case 1: Jika panjang target kombinasi adalah 0}

```
        if length = 0:
            return [[]] {Mengembalikan list yang berisi list kosong}

        {Base Case 2: Jika list input (nTuple) kosong}
        if Length(nTuple) = 0:
            return [] {Mengembalikan list kosong}

        result ← []

        for each index i from 0 to (Length(nTuple) - 1):

            {Slice array: ambil elemen mulai dari i+1 hingga akhir}
            sisa ← nTuple[i+1 ... ]

            {Panggilan rekursif: cari kombinasi dengan panjang dikurangi 1}
            listRekursif ← kombinasiBaris(sisa, length - 1)

            for each p in listRekursif:
                {Gabungkan elemen nTuple[i] dengan hasil rekursi}
                elemenBaru ← [nTuple[i]] + p
                result.append(elemenBaru)

        return result
End of Function
```

Func isValid (List of Tuple , List of List of Integer (matriks)) → Boolean {Fungsi ini akan mereturn true jika suatu solusi memenuhi seluruh aturan permainan Queens}
func isValidCombination(kombinasi, matriks):

```
        warnaTerpakai ← [] {list kosong}
        for each index i from 0 to (Length(kombinasi) – 1):
                curRow , curCol = kombinasi[i]
                {Aturan 1: Cek keunikan warna}
                if nilai matriks[baris][kolom] in warna_terpakai:
                        return False
                for each index  j from 0 to i - 1:
                        prevRow, prevCol ← kombinasi[j]

                        selisihBaris ← Abs(curBaris - prevBaris)
                        selisihKolom ← Abs(curKolom - prevKolom)

                        {Aturan 2: Cek tetangga diagonal (bersentuhan sudut)}
                        if selisihBaris = 1 and selisihKolom = 1:
                                return False

                        {Aturan 3: Cek baris atau kolom yang Sama}
                        if baris = prevBaris or kolom = prevKolom:
                                return False

                 warnaTerpakai.append(matriks[row][col])

            {Solusi memenuhi seluruh aturan}
            return True
End of Function
```

1.2.2 *Brute Force* Teroptimasi

Dalam versi yang teroptimasi kita akan memberikan batasan terhadap proses iterasi dalam pencarian kemumgkinan posisi peletakan ratu. Batasan yang kita berikan adalah jika suatu ratu diletakan pada suatu baris, maka ratu lain tidak boleh diletakan pada

baris yang sama dengan ratu tersebut. Hal ini akan mempercepat proses iterasi dalam menghasilkan seluruh kombinasi solusi. Teknik ini tidak tergolong dalam penyelesaian dengan metode heuristic dikarenakan, dari definisinya, teknik heuristik mungkin tidak selalu memberikan hasil optimal, tetapi secara ekstrim ia berguna pada penyelesaian persoalan. Hal ini tidak sesuai dengan metode optimasi yang dilakukan pada laporan ini, metode ini secara pasti, akan selalu memberi hasil yang optimal karena mengabaikan seluruh variasi di mana terdapat beberapa ratu yang berada pada baris yang sama. Selain itu, dalam versi ini, struktur data yang menyimpan seluruh kombinasi hanya berupa list satu dimensi, hal ini berbeda dengan versi awal yang membutuh list of tuple, sehingga versi ini menggunakan memori yang lebih kecil dibandingkan versi awal.

Pseudocode :

```
Func kombinasiBarisOptimal(List of Integer) → List of List of Integer
{Fungsi yang mereturn seluruh kombinasi koordinat untuk setiap ratu}
```

```
func kombinasiBarisOptimal(n):
    {Base Case: Jika panjang list n kurang dari atau sama dengan 1}
    if Length(n) <= 1:
        return [n]
    result ← []

    for each index i from 0 to (Length(n) - 1):
        current ← n[i]
    {List sisa dari gabungan elemen sebelum index i dan setelah index i}
        sisa ← n[0 ... i-1] + n[i+1 ... ]
        listRekursif ← kombinasiBarisOptimal(sisa)

        for each p in listRekursif:
            {Gabungkan elemen saat ini di depan hasil rekursi}
            elemenBaru ← [current] + p
```

```
                result.append(elemenBaru)

    return result
End of Function
```

```
func isValidCombinationOptimal(List of integer, List of list of Integer) → boolean
{Fungsi ini akan mereturn true jika suatu solusi memenuhi seluruh aturan
permainan Queens}
```

```
func isValidCombinationOptimal(combination, matriks):
    warnaTerpakai ← [] {list kosong}
    for each index i from 0 to (Length(combination) - 1):
        row ← i
        col ← combination[i]
        {Aturan 1: Cek keunikan warna}
        if matriks[row][col] in warnaTerpakai:
            return False

        for each index j from 0 to i - 1:
            prevRow ← j
            prevCol ← combination[j]

            selisihBaris ← Abs(row - prevRow)
            selisihKolom ← Abs(col - prevCol)

            {Aturan 2: Cek tetangga diagonal (bersentuhan sudut)}
            if selisihBaris = 1 and selisihKolom = 1:
                return False

        warnaTerpakai.append(matriks[row][col])
```

```
        return True
End of Function
```

BAB II
IMPLEMENTASI PROGRAM DENGAN BAHASA JAVA

Dalam tugas kecil ini, laporan ini menggunakan 3 file utama sebagai *source code*. Model.py berisi seluruh fungsi utama dalam menghasilkan seluruh kombinasi posisi ratu dan fungsi validasi, GUI.py berisi kelas *interface* dan fungsi-fungsi implementasi GUI untuk interaksi dengan user, serta main.py sebagai program utama untuk menjalankan keseluruhan proyek.

2.1 Model.py

Modul ini berisi fungs-fungsi utama dalam penyelesaian permainan *N-Queens* ini.

Nama	Tipe	Deskripsi
kombinasiBarisOptimal(n)	List of list of integer	Fungsi yang menghasilkan seluruh solusi *N-Queens* dengan penambahan *constraints*
kombinasiBarisOptimalGenerator(n)	List of list of integer	Fungsi yang menghasilkan satu persatu solusi *N-Queens* dengan penambahan *constraints*
kombinasiBaris(nTuple, length)	List of List of Tuple	Fungsi yang menghasilkan seluruh kombinasi *N-Queens* tanpa *constraints*
kombinasiBarisGenerator(nTuple, length)	List of List of Tuple	Fungsi yang menghasilkan satu persatu kombinasi *N-*

		Queens tanpa *constraints*
isPersegi(nRows, nCols)	Boolean	Fungsi yang memeriksa suatu puzzle N-Queens berbentuk persegi atau tidak.
isValidCombination(combination, matrix)	Boolean	Fungsi yang memeriksa salah satu variasi kombinasi dan memeriksa kebenarannya (versi brute force murni)
isValidCombinationOptimal(combination, matrix)	Boolean	Fungsi yang memeriksa salah satu variasi kombinasi dan memeriksa kebenarannya (versi brute force teroptimasi)
print_matrix(matrix, combination)	void	Fungsi yang print setiap kombinasi dalam bentuk matriks. (versi brute force murni)
print_matrix_optimal (matrix, combination=None)	Void	Fungsi yang print setiap kombinasi dalam bentuk matriks. (versi brute force teroptimasi)

generateRGB(char)	string	Fungsi yang memberikan hex coloer untuk setiap huruf.
isKotakSamaWarna(matrix)	Boolean	Fungsi yang memeriksa banyaknya warna unik sama dengan banyaknya baris/kolom pada papan.

Source code :

Model.py

```
import random
QUEENS_LABEL = "#"
def countWaranUnik(matrix):
    warnaUnik = set()
    for row in matrix:
        for color in row:
            warnaUnik.add(color)
    return len(warnaUnik)

def kombinasiBarisOptimal(n):
    if(len(n)<=1):
        return [n]
    result = []
    for i in range(len(n)):
        current = n[i]
        sisa = n[:i] + n[i+1:]
        for p in kombinasiBarisOptimal(sisa):
            result.append([current] + p)
```

```
    return result


def kombinasiBaris(nTuple, length):
    if length == 0:
        return [[]]
    if len(nTuple) == 0:
        return []
    result = []
    for i in range(len(nTuple)):
        sisa = nTuple[i+1:]
        for p in kombinasiBaris(sisa, length-1):
            result.append([nTuple[i]] + p)
    return result

def kombinasiBarisGenerator(nTuple, length, start=0):
    n = len(nTuple)
    if length == 0:
        yield []
        return
    if start >= n or n - start < length:
        return
    if length == 1:
        for i in range(start, n):
            yield [nTuple[i]]
        return
    for i in range(start, n - length + 1):
        current = nTuple[i]
        for suffix in kombinasiBarisGenerator(nTuple, length - 1, i + 1):
            result = [current]
            result.extend(suffix)
```

```
            yield result

    def kombinasiBarisOptimalGenerator(n):
        if len(n) <= 1:
            yield n
            return

        for i in range(len(n)):
            current = n[i]
            sisa = n[:i] + n[i+1:]
            for p in kombinasiBarisOptimalGenerator(sisa):
                yield [current] + p

    def hitungJumlahKombinasi(n, r):

        if r > n or r < 0:
            return 0
        if r == 0 or r == n:
            return 1

        r = min(r, n - r)

        result = 1
        for i in range(r):
            result = result * (n - i) // (i + 1)
        return result

    def isPersegi(nRows, nCols):
        return nRows == nCols

    def isValidCombinationOptimal(combination, matrix):
```

```
    usedColors = []
    for i in range(len(combination)):
        row = i
        col = combination[i]
        if matrix[row][col] in usedColors:
            return False
        for j in range(i):
            prev_row = j
            prev_col = combination[j]
            if abs(row - prev_row) == 1 and abs(col - prev_col) == 1:
                return False

        usedColors.append(matrix[row][col])
    return True

def isValidCombination(combination, matrix):
    n = len(combination)
    matrixSize = len(matrix)

    usedColors = set()
    usedRows = [False] * matrixSize
    usedCols = [False] * matrixSize

    i = 0
    while i < n:
        row, col = combination[i]

        if usedRows[row] or usedCols[col]:
            return False
```

```
color = matrix[row][col]
if color in usedColors:
    return False

if i > 0:
    pr0, pc0 = combination[0]
    rd0 = row - pr0
    cd0 = col - pc0
    if (rd0 == 1 or rd0 == -1) and (cd0 == 1 or cd0 == -1):
        return False

if i > 1:
    pr1, pc1 = combination[1]
    rd1 = row - pr1
    cd1 = col - pc1
    if (rd1 == 1 or rd1 == -1) and (cd1 == 1 or cd1 == -1):
        return False

j = 2
while j < i:
    prev_row, prev_col = combination[j]
    row_diff = row - prev_row
    col_diff = col - prev_col

    if (row_diff == 1 or row_diff == -1) and (col_diff == 1 or col_diff == -1):
        return False

    j += 1

usedRows[row] = True
usedCols[col] = True
```

```
            usedColors.add(color)

            i += 1

        return True



def print_matrix_optimal(matrix, combination=None):
    for i, row in enumerate(matrix):
        if combination is not None:
            row = list(row)
            row[combination[i]] = QUEENS_LABEL
        print(" ".join(row))

def print_matrix(matrix, combination=None):
    for i, row in enumerate(matrix):
        if combination is not None:
            row = list(row)
            for r, c in combination:
                if r == i:
                    row[c] = QUEENS_LABEL
        print(" ".join(row))

def generateRGB(char):
    colorMap = {
        'A': "#FF3434",
        'B': "#9D71A8",
        'C': '#45B7D1',
        'D': '#FFA07A',
        'E': '#98D8C8',
```

```
            'F': '#F7DC6F',
            'G': '#BB8FCE',
            'H': '#85C1E2',
            'I': '#F8B88B',
            'J': '#82E0AA',
            'K': '#F1948A',
            'L': '#AED6F1',
            'M': '#F5B041',
            'N': '#D7BDE2',
            'O': '#76D7C4',
            'P': '#F9E79F',
            'Q': '#F8B195',  #
            'R': '#C5E1A5',
            'S': '#FFD89B',
            'T': '#B19CD9',
            'U': '#80DEEA',
            'V': '#FF8A65',
            'W': '#81C784',
            'X': '#BA68C8',
            'Y': '#FFD54F',
            'Z': '#64B5F6',
        }
        return colorMap.get(char, '#CCCCCC')


    def isKotakSamaWarna(matrix):
        nWarnaUnik = countWaranUnik(matrix)
        nRows = len(matrix)
        if nWarnaUnik != nRows:
            return False
        return True
```

```
#============================================================
```

2.2 GUI.py

Modul ini berisi seluruh fungsi-fungsi untuk mengontrol tampilan UI ke layar pengguna.

- Attribut

Nama	Tipe	Deskripsi
Root	Tk	Main window untuk tkinter
Matrix	List of list of characters	Representasi papan *N-Queens*
Row	integer	Banyak baris
Col	integer	Banyak kolom
ukuranSel	integer	Ukuran satu sel
idRatu	List of ids	menyimpan ID dari semua object queen yang digambar di canvas.
isRunning	Boolean	Flag algoritma sedang berjalan
startTime	integer	Detik awal algori
currentIdx	integer	Indeks dari kombinasi yang sedang diproses.
Canvas	object	Tempat menggambar papan *N-Queens* .

- Method

Nama	Tipe	Deskripsi
drawGrid()	Void	Menggambarkan susunan papan.
saveSolutin()	File	Menyimpan solusi valid ke dalam txt.

drawQueen()	void	Menggambar simbol untuk ratu
saveCanvasAsImage()	File	Menyimpan canvas solusi sebagai png
startOptimal()	void	Memulai pencarian solusi versi brute force teroptimasi
startBrute()	void	Memulai pencarian solusi versi brute force murni
speedUpFunc()	void	Mempercepat waktu proses algoritma.
speedDownFunc()	void	Memperlambat waktu proses algoritma.
prosesAlgeo()	void	Bagian utama dalam menjalankan seluruh proses pencarian solusi permainan.

GUI.py

```
from PIL import Image, ImageDraw, ImageFont
from model import *


import tkinter as tk
from tkinter import Button, messagebox, filedialog
import time
import os
import threading
import queue


QUEENS_LABEL = "#"
```

```
class GUI:
    def __init__(self, root, matrix):
        self.root = root
        self.matrix = matrix
        self.rows = 0
        self.cols = 0
        self.ukuranSel = 64
        self.idRatu = []
        self.isRunning = False
        self.startTime = 0
        self.currentIdx = 0
        self.delay = 500
        self.mode = None
        self.root.title("Algoritma Brute Force untuk Masalah N-Queens")
        self.namaFileInput = ""

        self.workerThread = None
        self.resultQueue = queue.Queue()
        self.stopFlag = threading.Event()
        self.updateInterval = 1000000
        self.progressCounter = 0

        topF = tk.Frame(root)
        topF.pack(pady=5, padx=10)

        cF = tk.Frame(root)
        cF.pack(pady=10,padx=10)

        self.startOptimalButton = tk.Button(cF, text="Optimal",bg="red",
fg="white", font=("Arial", 12, "bold"), command=self.startOptimal)
        self.startOptimalButton.pack(side=tk.LEFT, padx=5)
```

```
        self.startBruteButton = tk.Button(cF, text="Brute Force",bg="purple",
fg="white", font=("Arial", 12, "bold"), command=self.startBrute)
        self.startBruteButton.pack(side=tk.LEFT, padx=5)

        self.timer = tk.Label(cF, text="Waktu: 0.000dtk", font=("Arial", 12))
        self.timer.pack(side=tk.LEFT, padx=5)

        self.speedLabel = tk.Label(cF, text=f"Delay: {self.delay}ms",
font=("Arial", 12))
        self.speedLabel.pack(side=tk.LEFT, padx=5)

        self.speedUp = tk.Button(cF, text="Speed Up", bg="#4CAF50",
fg="white", font=("Arial", 12, "bold"), command=self.speedUpFunc)
        self.speedUp.pack(side=tk.LEFT, padx=5)

        self.speedDown = tk.Button(cF, text="Speed Down", bg="#ff0000",
fg="white", font=("Arial", 12, "bold"), command=self.speedDownFunc)
        self.speedDown.pack(side=tk.LEFT, padx=5)

        self.status = tk.Label(root, text="Status: Ready", font=("Arial", 12))
        self.status.pack(pady=10)

        self.loadFileButton = tk.Button(root, text="Load File", bg="#607D8B",
fg="white", font=("Arial", 12, "bold"), command=self.loadFile)
        self.loadFileButton.pack(pady=5)

        self.fileLabel = tk.Label(root, text="File: (belum dipilih)", font=("Arial",
10))
        self.fileLabel.pack(padx=5)
```

```
        self.saveSolutionButton = tk.Button(root, text="Simpan As text",
bg="#2196F3", fg="white", font=("Arial", 12, "bold"),
command=self.saveSolution)
        self.saveSolutionButton.pack(pady=10)

        self.saveCanvasButton = tk.Button(root, text="Simpan As Image",
bg="#002A8D", fg="white", font=("Arial", 12, "bold"),
command=self.saveCanvasAsImage)
        self.saveCanvasButton.pack(pady=5)

        self.canvas = tk.Canvas(root, background="white",
width=max(self.cols,1)*self.ukuranSel,
height=max(self.rows,1)*self.ukuranSel)
        self.canvas.pack(padx=20, pady=20)

        if self.rows > 0:
            self.drawGrid()

    def loadFile(self):
        if self.isRunning:
            messagebox.showwarning("Peringatan", "Algoritma sedang berjalan!")
            return
        filepath = filedialog.askopenfilename(
            title="Pilih file papan",
            filetypes=[("Text files", "*.txt"), ("All files", "*.*")]
        )

        if not filepath:
            return
        try:
            newMatrix = []
```

```
            colors = [chr(i).upper() for i in range(ord('a'), ord('z') + 1)]
            with open(filepath, "r") as file:
                for line in file:
                    line = line.strip()
                    if line:
                        if any(c not in colors for c in line):
                            messagebox.showerror("Error", f"Karakter tidak valid:
{line}")
                            return
                        newMatrix.append(list(line))
            if len(newMatrix) == 0:
                messagebox.showerror("Error", "File kosong!")
                return
            self.matrix = newMatrix
            self.namaFileInput = os.path.basename(filepath)
            self.rows = len(newMatrix)
            self.cols = len(newMatrix[0])
            for row in newMatrix:
                if len(row) != self.cols:
                    messagebox.showerror("Error", "Semua baris harus memiliki
jumlah kolom yang sama!")
                    return


            self.idRatu.clear()
            self.currentIdx = 0
            self.canvas.config(width=self.cols*self.ukuranSel,
height=self.rows*self.ukuranSel)
            self.canvas.delete("all")
            self.drawGrid()
            self.fileLabel.config(text=f"File: {self.namaFileInput}")
            self.status.config(text="Status: Ready", fg="black")
```

```
            self.timer.config(text="Waktu: 0.000dtk")
        except Exception as e:
            messagebox.showerror("Error", f"Gagal membaca file: {e}")
    def drawGrid(self):
        for i in range(self.rows):
            for j in range(self.cols):
                x1 = j * self.ukuranSel
                y1 = i * self.ukuranSel
                x2 = x1 + self.ukuranSel
                y2 = y1 + self.ukuranSel
                char_color = self.matrix[i][j]
                color_hex = generateRGB(char_color)
                self.canvas.create_rectangle(x1, y1, x2, y2, fill=color_hex,
outline="black")
                self.canvas.create_text(x1 + 10, y1 + 10, text=char_color,
font=("Arial", 8), fill="#333")

    def saveSolution(self):

        if not hasattr(self, 'aktivCara'):
            messagebox.showwarning("Peringatan", "Tidak ada solusi yang sedang
ditampilkan untuk disimpan!")
            return

        if self.isRunning:
            messagebox.showwarning("Peringatan", "Tidak dapat menyimpan saat
algoritma sedang berjalan!")
            return

        if self.currentIdx >= len(self.aktivCara):
```

```
            messagebox.showwarning("Peringatan", "Tidak ada solusi yang sedang
ditampilkan untuk disimpan!")
            return

        curDir = os.path.dirname(os.path.abspath(__file__))
        rootDir = os.path.dirname(curDir)
        testDir = os.path.join(rootDir, "test")
        if not os.path.exists(testDir):
            os.makedirs(testDir)

        curComb = self.aktivCara[self.currentIdx]
        isValid = isValidCombinationOptimal(curComb, self.matrix) if self.mode
== 'optimal' else isValidCombination(curComb, self.matrix)
        if isValid:
            filename = f"solution_{self.namaFileInput}"
            filepath = os.path.join(testDir, filename)
            with open(filepath, "w") as file:
                for i in range(self.rows):
                    row = list(self.matrix[i])
                    if self.mode == 'optimal':
                        row[curComb[i]] = QUEENS_LABEL
                    else:
                        for r, c in curComb:
                            if r == i:
                                row[c] = QUEENS_LABEL
                    file.write("".join(row) + "\n")
            messagebox.showinfo("Sukses", f"Solusi disimpan sebagai
test/{filename}")
        else:
            messagebox.showwarning("Peringatan", "Kombinasi saat ini bukan
solusi yang valid!")
```

```
    def drawQueensOptimal(self, combination):
        for idRatu in self.idRatu:
            self.canvas.delete(idRatu)
        self.idRatu.clear()

        for row, col in enumerate(combination):
            x = col * self.ukuranSel + self.ukuranSel / 2
            y = row * self.ukuranSel + self.ukuranSel / 2
            qId = self.canvas.create_oval(x-15, y-15, x+15, y+15, fill="black",
outline="white", width=2)
            qId_txt = self.canvas.create_text(x, y, text=QUEENS_LABEL,
fill="white", font=("Arial", 12, "bold"))
            self.idRatu.append(qId)
            self.idRatu.append(qId_txt)

    def drawQueens(self, combination):
        for idRatu in self.idRatu:
            self.canvas.delete(idRatu)
        self.idRatu.clear()

        for row, col in combination:
            x = col * self.ukuranSel + self.ukuranSel / 2
            y = row * self.ukuranSel + self.ukuranSel / 2
            qId = self.canvas.create_oval(x-15, y-15, x+15, y+15, fill="black",
outline="white", width=2)
            qId_txt = self.canvas.create_text(x, y, text=QUEENS_LABEL,
fill="white", font=("Arial", 12, "bold"))
            self.idRatu.append(qId)
            self.idRatu.append(qId_txt)
```

```
    def saveCanvasAsImage(self):
        if not hasattr(self, 'aktivCara') or self.currentIdx >= len(self.aktivCara):
            messagebox.showwarning("Peringatan", "Tidak ada solusi yang sedang
ditampilkan untuk disimpan!")
            return

        if self.isRunning:
            messagebox.showwarning("Peringatan", "Tidak dapat menyimpan saat
algoritma sedang berjalan!")
            return

        curPath = os.path.dirname(os.path.abspath(__file__))
        rootDir = os.path.dirname(curPath)
        testDir = os.path.join(rootDir, "test")
        if not os.path.exists(testDir):
            os.makedirs(testDir)

        width = self.cols * self.ukuranSel
        height = self.rows * self.ukuranSel
        img = Image.new('RGB', (width, height), 'white')
        draw = ImageDraw.Draw(img)
        for i in range(self.rows):
            for j in range(self.cols):
                x1 = j * self.ukuranSel
                y1 = i * self.ukuranSel
                x2 = x1 + self.ukuranSel
                y2 = y1 + self.ukuranSel
                char_color = self.matrix[i][j]
                color_hex = generateRGB(char_color)
                draw.rectangle([x1, y1, x2, y2], fill=color_hex, outline="black")
```

```
                draw.text((x1 + 5, y1 + 5), char_color, fill="#333")



        if hasattr(self, 'aktivCara') and self.currentIdx < len(self.aktivCara):
            curComb = self.aktivCara[self.currentIdx]
            if self.mode == 'optimal':
                positions = [(i, curComb[i]) for i in range(len(curComb))]
            else:
                positions = [(r, c) for r, c in curComb]
            for row, col in positions:
                cx = col * self.ukuranSel + self.ukuranSel // 2
                cy = row * self.ukuranSel + self.ukuranSel // 2
                r = 15
                draw.ellipse([cx-r, cy-r, cx+r, cy+r], fill="black", outline="white",
width=2)
                draw.text((cx-4, cy-6), QUEENS_LABEL, fill="white")
        namaFile = os.path.splitext(self.namaFileInput)[0]
        filename = f"solusi_{namaFile}.png"
        filepath = os.path.join(testDir, filename)
        img.save(filepath, 'PNG')

        messagebox.showinfo("Sukses", f"Canvas disimpan sebagai
test/{filename}")

    def startOptimal(self):
        if len(self.matrix) == 0:
            messagebox.showwarning("Peringatan", "Silakan load file papan
terlebih dahulu!")
            return
        if self.isRunning:
```

```
            return

        if not isKotakSamaWarna(self.matrix):
            messagebox.showerror("Error", "Jumlah warna unik pada papan tidak
sama dengan jumlah kotak!")
            return

        self.mode = 'optimal'
        self.aktivCara = kombinasiBarisOptimal([i for i in range(self.cols)])
        self.isRunning = True
        self.startTime = time.time()
        self.currentIdx = 0
        self.startOptimalButton.config(state=tk.DISABLED)
        self.startBruteButton.config(state=tk.DISABLED)
        self.prosesAlgo()

    def workerBruteForce(self):
        try:
            allPositions = [(r, c) for r in range(self.rows) for c in range(self.cols)]
            generator = kombinasiBarisGenerator(allPositions, self.cols)

            iterationCount = 0
            for combination in generator:
                if self.stopFlag.is_set():
                    self.resultQueue.put(('stopped', None, iterationCount))
                    return

                iterationCount += 1

                if iterationCount % self.updateInterval == 0:
                    self.resultQueue.put(('progress', combination, iterationCount))
```

```
                if isValidCombination(combination, self.matrix):
                    self.resultQueue.put(('found', combination, iterationCount))
                    return

            self.resultQueue.put(('notfound', None, iterationCount))

        except Exception as e:
            self.resultQueue.put(('error', str(e), 0))

    def startBrute(self):
        if len(self.matrix) == 0:
            messagebox.showwarning("Peringatan", "Silakan load file papan
terlebih dahulu!")
            return
        if self.isRunning:
            return
        if not isKotakSamaWarna(self.matrix):
             messagebox.showerror("Error", "Jumlah warna unik pada papan tidak
sama dengan jumlah kotak!")
             return

        if not isPersegi(self.rows, self.cols):
            messagebox.showerror("Error", "Papan harus berbentuk persegi!")
            return

        self.mode = 'bruteforce'
        self.isRunning = True
        self.startTime = time.time()
        self.progressCounter = 0
        self.stopFlag.clear()
```

```
        while not self.resultQueue.empty():
            self.resultQueue.get()

        self.startOptimalButton.config(state=tk.DISABLED)
        self.startBruteButton.config(state=tk.DISABLED)

        self.workerThread = threading.Thread(target=self.workerBruteForce,
daemon=True)
        self.workerThread.start()

        self.checkQueue()

    def checkQueue(self):

        try:
            msgType, data, iterCount = self.resultQueue.get_nowait()

            if msgType == 'progress':
                self.progressCounter = iterCount
                waktu = time.time() - self.startTime
                self.timer.config(text=f"Waktu: {waktu:.3f}dtk")
                self.status.config(text=f"Testing iteration: {iterCount:,} | Last:
{data}", fg="black")
                self.drawQueens(data)

            elif msgType == 'found':
                self.isRunning = False
                waktu = time.time() - self.startTime
                self.timer.config(text=f"Waktu: {waktu:.3f}dtk")
                self.status.config(text=f"Solusi ditemukan: {data}", fg="green")
```

```
                self.drawQueens(data)

                self.aktivCara = [data]
                self.currentIdx = 0

                messagebox.showinfo("Sukses", f"Solusi Ditemukan!\n\nIterasi:
{iterCount:,}\nWaktu: {waktu:.3f} detik")
                self.startOptimalButton.config(state=tk.NORMAL)
                self.startBruteButton.config(state=tk.NORMAL)
                return

            elif msgType == 'notfound':
                self.isRunning = False
                waktu = time.time() - self.startTime
                self.timer.config(text=f"Waktu: {waktu:.3f}dtk")
                self.status.config(text=f"Tidak ada solusi (checked {iterCount:,}
combinations)", fg="red")
                messagebox.showinfo("Info", f"Tidak ada solusi yang valid\n\nTotal
iterasi: {iterCount:,}\nWaktu: {waktu:.3f} detik")
                self.startOptimalButton.config(state=tk.NORMAL)
                self.startBruteButton.config(state=tk.NORMAL)
                return

            elif msgType == 'stopped':
                self.isRunning = False
                self.status.config(text="Dihentikan oleh user", fg="orange")
                self.startOptimalButton.config(state=tk.NORMAL)
                self.startBruteButton.config(state=tk.NORMAL)
                return

            elif msgType == 'error':
```

```
                self.isRunning = False
                self.status.config(text=f"Error: {data}", fg="red")
                messagebox.showerror("Error", f"Terjadi error: {data}")
                self.startOptimalButton.config(state=tk.NORMAL)
                self.startBruteButton.config(state=tk.NORMAL)
                return

            except queue.Empty:
                if self.isRunning:
                    waktu = time.time() - self.startTime
                    self.timer.config(text=f"Waktu: {waktu:.3f}dtk")

            if self.isRunning:
                self.root.after(100, self.checkQueue)

        def speedUpFunc(self):
            self.delay = int(max(1, self.delay * 0.5))
            self.speedLabel.config(text=f"Delay: {self.delay}ms")

        def speedDownFunc(self):
            newDelay = int(self.delay * 1.5)

            if newDelay <= self.delay:
                self.delay += 1
            else :
                self.delay = newDelay
            self.speedLabel.config(text=f"Delay: {self.delay}ms")

        def prosesAlgo(self):
```

```
            if self.currentIdx >= len(self.aktivCara):
                self.isRunning = False
                self.startOptimalButton.config(state=tk.NORMAL)
                self.startBruteButton.config(state=tk.NORMAL)
                messagebox.showinfo("Info", "Tidak ada solusi yang valid")
                self.status.config(text="Selesai")
                return

            if not isPersegi(self.rows, self.cols):
                self.isRunning = False
                self.startOptimalButton.config(state=tk.NORMAL)
                self.startBruteButton.config(state=tk.NORMAL)
                messagebox.showerror("Error", "Papan harus berbentuk persegi!")
                self.status.config(text="Papan tidak persegi")
                return

            curComb = self.aktivCara[self.currentIdx]
            waktu = time.time() - self.startTime
            self.timer.config(text=f"Waktu: {waktu:.3f}dtk")
            self.status.config(text=f"Kombinasi: {curComb}", fg="black")

            if self.mode == 'optimal':
                self.drawQueensOptimal(curComb)
                isValid = isValidCombinationOptimal(curComb, self.matrix)
            else:
                self.drawQueens(curComb)
                isValid = isValidCombination(curComb, self.matrix)

            if isValid:
                self.isRunning = False
                self.status.config(text=f"Solusi ditemukan: {curComb}", fg="black")
```

```
                if self.mode == 'optimal':
                    self.drawQueensOptimal(curComb)
                else:
                    self.drawQueens(curComb)
                messagebox.showinfo("Sukses", f"Solusi Ditemukan dalam
        {time.time() - self.startTime:.3f} detik")
                self.startOptimalButton.config(state=tk.NORMAL)
                self.startBruteButton.config(state=tk.NORMAL)
                return

            self.currentIdx += 1
            self.root.after(self.delay, self.prosesAlgo)
```

2.3 Main.py

Bagian bootstrapper untuk program utama.

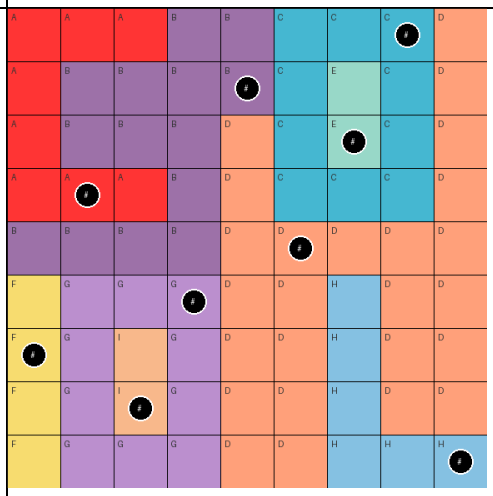
Main.py

```
from GUI import GUI
import tkinter as tk

QUEENS_LABEL = "#"

root = tk.Tk()
app = GUI(root, [])
root.mainloop()
```

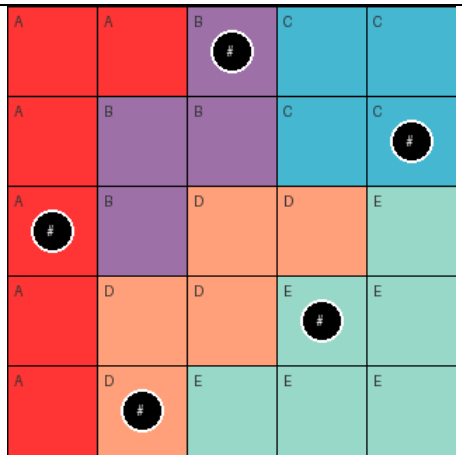
BAB III
SOURCE CODE DAN STRUKTUR PROGRAM

3.1 Output:

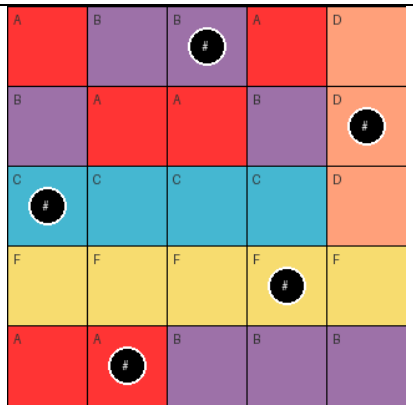
3.1.1 Test 1

Input Text	Output Text	Output Image
AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH	AAABBCC#D ABBB#CECD ABBBDC#CD A#ABDCCCD BBBBD#DDD FGG#DDHDD #GIGDDHDD FG#GDDHDD FGGGDDHH#	

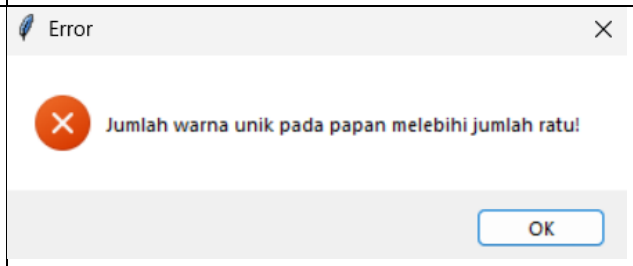
3.1.2 Test 2

Input Text	Output Text	Output Image
AABCC ABBCC ABDDE ADDEE ADEEE	AA#CC ABBC# #BDDE ADD#E A#EEE	

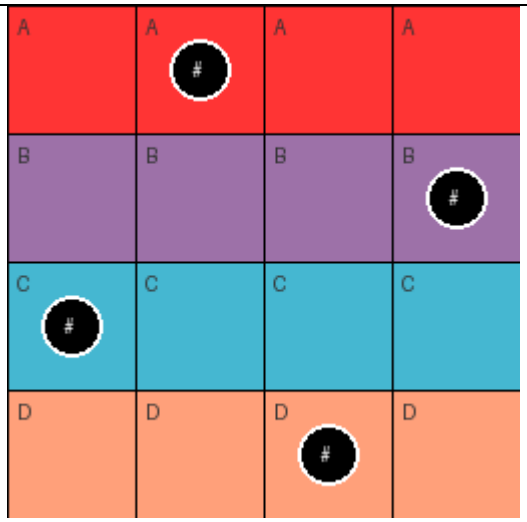
3.1.3 Test 3

Input Text	Output Text	Output Image
ABBAD BAABD CCCCD FFFFF AABBB	AB#AD BAAB# #CCCD FFF#F A#BBB	

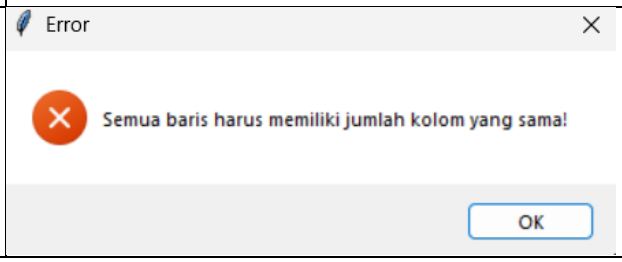
3.1.4 Test 4

Input Teks	Eror Output
AAAA BBBB CCCC DDDE	

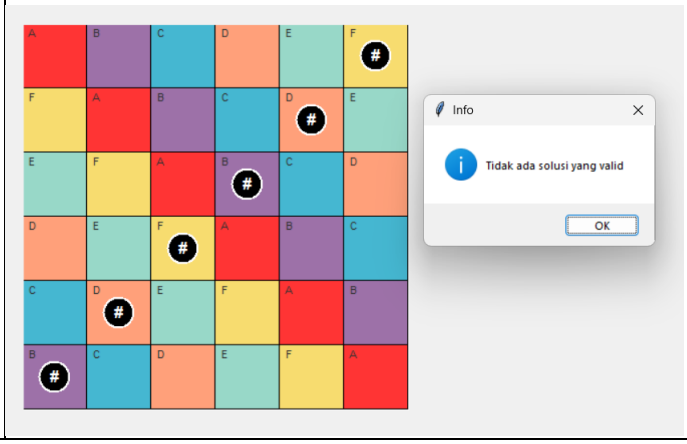
3.1.5 Test 5

Input Text	Output Text	Output Image
AAAA BBBB CCCC DDDD	A#AA BBB# #CCC DD#D	

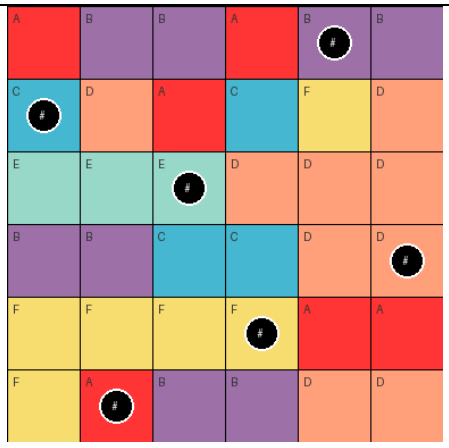
3.1.6 Test 6

Input Teks	Eror Output
AABB CCDDF GGHH	

3.1.7 Test 7

Input Teks	Eror Output
ABCDEF FABCDE EFABCD DEFABC CDEFAB BCDEFA	

3.1.8 Test 8

Input Text	Output Text	Output Image
ABBABB CDACFD EEEDDD BBCCDD FFFFAA FABBDD	ABBA#B #DACFD EE#DDD BBCCD# FFF#AA F#BBDD	

LAMPIRAN

Tautan:

https://github.com/Kurt-Mikhael/Tucil1_13524065

SURAT PERNYATAAN

Pernyataan Tidak Melakukan Kecurangan:

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

[Kurt Mikhael Purba]