

LAPORAN TUGAS KECIL
STRATEGI ALGORITMA (IF2211)
PROGRAM “PENYELESAIAN PERMAINAN QUEEN LINKEDIN”



Disusun Oleh:

Angelina Andra Alanna

(13524079)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG

2025

BAB 1

ALGORITMA

1.1. Deskripsi Permasalahan

Tugas Kecil 1 ini memiliki peraturan yaitu untuk menentukan N jumlah Queens pada papan berukuran $n \times n$. Papan terbagi menjadi wilayah yang ditandai dengan huruf kapital yang berbeda. Penempatan Queen harus memenuhi empat aturan sekaligus:

Papan permainan terdiri dari beberapa wilayah dengan warna berbeda (ditandai dengan huruf A-Z), dan setiap wilayah hanya boleh berisi maksimal satu queen.

1.2. Pendekatan Algoritma Brute Force

Algoritma yang digunakan adalah brute force murni. Program mencoba semua kemungkinan penempatan N queens pada papan $N \times N$ tanpa asumsi atau optimasi apapun. Setiap queen bebas ditempatkan di kolom mana saja (0 sampai N-1) untuk setiap baris, menghasilkan total N^N kombinasi yang harus diperiksa satu per satu.

Tidak ada teknik heuristik, pruning, atau eliminasi dini yang digunakan. Semua N^N kombinasi dibangkitkan terlebih dahulu, kemudian tiap kombinasi diperiksa apakah memenuhi seluruh constraint. Ini adalah pendekatan brute force yang paling murni karena benar-benar mencoba setiap kemungkinan yang ada.

1.3. Tahapan Algoritma

1.3.1. Inisialisasi

Program dimulai dengan membaca input dari file .txt yang berisi konfigurasi papan.

Format input:

```
AABBCCDD
AABBCCDD
EEFFGGHH
EEFFGGHH
...
```

Setiap huruf merepresentasikan region warna yang berbeda. Program kemudian:

- Membuat representasi internal papan sebagai matriks 2 dimensi
- Membangun peta wilayah warna (map_warna) yang menyimpan koordinat setiap sel untuk setiap warna
- Menginisialisasi struktur data untuk tracking queen yang sudah ditempatkan

1.3.2. Validasi Awal

Sebelum mulai nyari solusi, program ngelakuin beberapa pengecekan dulu buat mastiin board-nya valid dan bisa di-solve.

- **Validasi Ukuran Papan**

- Semua baris panjangnya sama (misal kalau board 8×8 , tiap baris harus ada 8 karakter)
- Semua isi papan cuma boleh huruf kapital A-Z (gak boleh ada angka, spasi, atau karakter aneh)

Kalau ada yang salah, program langsung memberitahu letak error-nya. Misalnya "Row 3 punya panjang 7, seharusnya 8" atau "Ada karakter invalid di posisi (2, 3)".

- **Validasi Warna Papan**

Ini yang penting buat prediksi bisa diselesaikan atau tidak. Program menghitung:

- Berapa banyak warna unik yang ada di papan (misal board 8×8 punya 15 warna berbeda: A, B, C, sampai O)
- Berapa kali tiap warna muncul di papan

Terus program simpen informasi warna ini dalam dua cara:

1. **Set warna unik** - Kumpulan semua warna yang ada, jadi gampang tau ada berapa warna total
2. **Dictionary lokasi warna** - Nyimpen di mana aja posisi tiap warna.

Contohnya:

```
{
  'A': [(0,0), (0,1), (1,0), (1,1)], # Warna A ada di 4 posisi ini
  'B': [(0,2), (0,3), (1,2), (1,3)], # Warna B ada di 4 posisi ini
  'C': [(2,0), (2,1), (3,0), (3,1)], # Warna C ada di 4 posisi ini
}
```

Program kasih warning ke user kalau misalnya jelas-jelas gak mungkin ada solusi (warna kurang dari jumlah queens), tapi tetap kasih opsi buat lanjut kalau user mau coba aja.

Ini semua ngebantu user buat tau dari awal apakah puzzle-nya reasonable atau enggak sebelum nunggu program solving yang bisa lama.

1.3.3. Penjelasan Kombinasi

Program membangkitkan semua N^N kombinasi menggunakan fungsi rekursif `_semua_kombinasi(panjang)`. Cara kerjanya: untuk setiap sub-kombinasi sepanjang $(panjang-1)$, ditambahkan setiap kemungkinan kolom $(0$ sampai $N-1$) di posisi terakhir
Cara kerjanya:

```
def _semua_kombinasi(self, panjang):
    if panjang == 0:
        return [[]]          # Base case: satu kombinasi
    kosong
    hasil = []
    for sub in self._semua_kombinasi(panjang - 1):
        for col in range(self.n):    # Tiap kolom 0..N-1
            hasil.append(sub + [col])
    return hasil
```

1.3.4. Constraint Checking

Fungsi `_bisa_lanjut()` memeriksa apakah penempatan queen valid:

```
def _bisa_lanjut(row, col, posisi_queen):
    warna = papan_awal[row][col]

    for queen_row, queen_col in posisi_queen:
        # Constraint 1 & 2: Row dan Column
        if queen_row == row or queen_col == col:
            return False

        # Constraint 3: Adjacent (8 sel sekitar)
        if abs(queen_row - row) <= 1 and abs(queen_col - col) <= 1:
            return False

        # Constraint 4: Warna yang sama
        if papan_awal[queen_row][queen_col] == warna:
            return False

    return True
```

Karena permutasi sudah menjamin tidak ada queen di baris yang sama, constraint baris tidak perlu dicek lagi. Tiga constraint yang dicek adalah: kolom sama, bertetangga (jarak ≤ 1 ke segala arah termasuk diagonal), dan region warna sama.

1.3.5. Visualisasi untuk GUI

Untuk keperluan GUI dengan animasi live, program menggunakan `solve_visual_callback()` yang mengirimkan update ke GUI setiap kali permutasi baru dicoba. Setiap permutasi, semua N queens langsung ditampilkan sekaligus pada posisi yang sesuai, lalu diganti dengan permutasi berikutnya. Ini memperlihatkan proses brute force secara nyata: setiap kombinasi dicoba satu per satu sampai ketemu yang valid.

1.3.6. Flow Lengkap Algoritma

Berikut alur lengkap algoritma dari awal sampai akhir:

- Input dibaca dari file .txt dan divalidasi (ukuran dan karakter)
- Analisis warna: buat set warna unik dan dictionary lokasi per warna

- Bangkitkan semua N^N kombinasi menggunakan `_semua_kombinasi(N)`
- Untuk setiap kombinasi: tambah jumlah_kasus, cek semua constraint
- Kalau valid: bangun papan solusi, catat waktu dan kasus, return HasilSolusi(found=True)
- Kalau semua N^N kombinasi habis dan tidak ada yang valid: return HasilSolusi(found=False)

1.4. Kompleksitas Algoritma

Kompleksitas Waktu: $O(N^N \times N^2)$. Ada N^N kombinasi yang dibangkitkan dan diperiksa, dan setiap pemeriksaan butuh $O(N^2)$ untuk mengecek semua pasangan queen.

Kompleksitas Ruang: $O(N^N \times N)$ untuk menyimpan semua kombinasi sekaligus, plus $O(N^2)$ untuk papan.

Konsekuensi praktisnya sangat terasa seiring bertambahnya ukuran papan:

- $N=4$: $4^4 = 256$ kombinasi \rightarrow sangat cepat
- $N=8$: $8^8 = 16.777.216$ kombinasi \rightarrow sangat lambat (beberapa menit)

Ini sangat wajar untuk brute force murni yang tidak melakukan optimasi apapun. Kalau tidak ada solusi, program benar-benar mencoba semua N^N kombinasi sampai habis.

BAB 2

SOURCE PROGRAM

2.1. gui_visual.py

```
import os
import sys
import tkinter as tk
from tkinter import ttk, filedialog, messagebox, scrolledtext
from threading import Thread
import queue
import time
from typing import Optional
from PIL import Image, ImageDraw, ImageFont
from io import BytesIO
import subprocess, tempfile, os

sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))

from solver_core import HasilSolusi, SolverUtama,
validasi_warna_papan, validasi_ukuran_papan, cek_konek_warna
from io_code import baca_file_papan, tulis_file_solusi, cek_folder

class QueenGUIVisual:
    COLORS = {
        'A': '#E57373', 'B': '#F06292', 'C': '#BA68C8', 'D':
        '#9575CD',
        'E': '#7986CB', 'F': '#64B5F6', 'G': '#4FC3F7', 'H':
        '#4DD0E1',
        'I': '#4DB6AC', 'J': '#81C784', 'K': '#AED581', 'L':
        '#DCE775',
        'M': '#FFF176', 'N': '#FFD54F', 'O': '#FFB74D', 'P':
        '#FF8A65',
        'Q': '#A1887F', 'R': '#90A4AE', 'S': '#F48FB1', 'T':
        '#CE93D8',
        'U': '#B39DDB', 'V': '#9FA8DA', 'W': '#80DEEA', 'X':
        '#A5D6A7',
        'Y': '#FFE082', 'Z': '#FFAB91'
```

```

    }

    def __init__(self, root):
        self.root = root
        self.root.title("Penyelesaian Queens Berwarna + Animasi")
        self.root.geometry("1200x800") # formatnya
        ("lebarxtinggi+x+y") -> x + y itu kek lokasi
        root.minsize(1000,700)

        self.papan = None
        self.n = 0
        self.solver = None
        self.solver_thread = None
        self.is_solving = False
        self.animation_speed = 0.05

        self.update_queue = queue.Queue()

        self.cell_size = 50
        self.canvas_cells = {}

        self._create_widgets()
        self._perbaiki_layout()

        self._process_queue()

    def _create_widgets(self):
        self.title_label = tk.Label(
            self.root,
            text = "Penyelesaian Queens Warna + Animasi",
            font = ("Arial", 16, "bold"),
            pady = 10
        )

        self.left_panel = ttk.Frame(self.root)
        self.input_frame = ttk.LabelFrame(self.left_panel, text =
"Input", padding = 10)

        self.select_file_btn = ttk.Button(

```

```

        self.input_frame,
        text = "Pilih File Papan (.txt)",
        command=self._select_file
    )

    self.file_label = tk.Label(
        self.input_frame,
        text="Tidak ada file yang dipilih",
        fg = "gray",
        wraplength=250
    )

    self.info_frame = ttk.LabelFrame(self.left_panel, text =
    "Info Papan")

    self.info_text = tk.Text(
        self.info_frame,
        width = 35,
        height = 8,
        font = ("Arial", 9),
        wrap = tk.WORD
    )

    self.info_text.config(state=tk.DISABLED)

    self.speed_frame = ttk.LabelFrame(self.left_panel, text =
    "Kecepatan Animasi", padding = 10)

    self.speed_value_label = tk.Label(
        self.speed_frame,
        text = "Sedang",
        font = ("Arial", 9, "bold")
    )

    self.scale_kecepatan = ttk.Scale(
        self.speed_frame,
        from_ = 100,
        to = 1,
        orient = tk.HORIZONTAL,

```



```

        command = self._update_speed
    )

    self.scale_kecepatan.set(25)
    self.control_frame = ttk.LabelFrame(self.left_panel,
text="Controls", padding = 10)

    self.solve_btn = ttk.Button(
        self.control_frame,
        text = "Solve dengan Animasi",
        command = self._start_solving,
        state = tk.DISABLED,
    )

    self.save_btn = ttk.Button(
        self.control_frame,
        text = "Save Solusi",
        command =self._save_solution,
        state = tk.DISABLED
    )

    self.stats_frame = ttk.LabelFrame(self.left_panel,
text="Statistik", padding = 10)

    self.stats_text = tk.Text(
        self.stats_frame,
        width = 35,
        height = 6,
        font = ("Courier", 9),
        wrap = tk.WORD
    )

    self.stats_text.config(state=tk.DISABLED)

    self.right_panel = ttk.Frame(self.root)

    self.canvas_frame = ttk.LabelFrame(self.right_panel,
text="Papan Visualisasi Real-Time", padding = 10)

```

```

self.canvas_container = ttk.Frame(self.canvas_frame)

self.canvas = tk.Canvas(
    self.canvas_container,
    bg = "white",
)

self.v_scrollbar = ttk.Scrollbar(
    self.canvas_container,
    orient=tk.VERTICAL,
    command = self.canvas.yview
)

self.h_scrollbar = ttk.Scrollbar(
    self.canvas_container,
    orient=tk.HORIZONTAL,
    command=self.canvas.xview
)

self.canvas.configure(
    yscrollcommand = self.v_scrollbar.set,
    xscrollcommand = self.h_scrollbar.set
)

self.status_label = tk.Label(
    self.right_panel,
    text = "Ready",
    font = ("Arial", 10),
    anchor =tk.W,
    pady = 5
)

self.legend_a_frame = ttk.LabelFrame(self.right_panel, text
= "Legenda", padding=5)
self.legend_a_text = tk.Label(self.legend_a_frame,
                                text = "# = Tempat Ratu | □ =
Kotak Kosong | Warna = Pembeda Wilayah",

```

```

        font = ("Arial", 9),
        justify=tk.LEFT
    )

    def _perbaiki_layout(self):
        self.title_label.grid(row=0, column=0, columnspan=2,
pady=10, sticky=tk.EW)

        self.left_panel.grid(row=1, column=0, padx=10, pady=5,
sticky=tk.NSEW)

        self.input_frame.pack(fill=tk.X, pady=5)
        self.select_file_btn.pack(pady=5)
        self.file_label.pack(pady=5)

        self.info_frame.pack(fill=tk.BOTH, expand=True, pady=5)
        self.info_text.pack(fill=tk.BOTH, expand=True)

        self.speed_frame.pack(fill=tk.X, pady=5)
        self.speed_value_label.pack()
        self.scale_kecepatan.pack(fill=tk.X, pady=5)
        self.speed_value_label.pack()

        self.control_frame.pack(fill=tk.X, pady=5)
        self.solve_btn.pack(fill=tk.X, pady=2)
        self.save_btn.pack(fill=tk.X, pady=2)

        self.stats_frame.pack(fill=tk.BOTH, expand=True, pady=5)
        self.stats_text.pack(fill=tk.BOTH, expand=True)

        self.right_panel.grid(row=1, column=1, padx=10, pady=5,
sticky=tk.NSEW)

        self.canvas_frame.pack(fill=tk.BOTH, expand=True, pady=5)
        self.canvas_container.pack(fill=tk.BOTH, expand=True)

        self.canvas.grid(row=0, column=0, sticky=tk.NSEW)
        self.v_scrollbar.grid(row=0, column=1, sticky=tk.NS)
        self.h_scrollbar.grid(row=1, column=0, sticky=tk.EW)

```

```

self.canvas_container.grid_rowconfigure(0, weight=1)
self.canvas_container.grid_columnconfigure(0, weight=1)

self.legend_frame.pack(fill=tk.X, pady=5)
self.legend_text.pack()

self.status_label.pack(fill=tk.X, pady=2)

self.root.columnconfigure(0, weight=1)
self.root.columnconfigure(1, weight=3)
self.root.rowconfigure(1, weight=1)

def _select_file(self):
    filepath = filedialog.askopenfilename(
        title = "Pilih File",
        initialdir="../test",
        filetypes=[("Text Files", "*.txt"), ("All Files",
        " *.*") ]
    )

    if not filepath:
        return

    try:
        self.papan, self.n = baca_file_papan(filepath)

        self.cur_file = filepath
        is_valid, error_message =
validasi_ukuran_papan(self.papan, self.n)
        koneksi_valid, warna_terpisah =
cek_konek_warna(self.papan)
        if not koneksi_valid:
            messagebox.showerror("Papan Tidak Valid", "Ada
Warna yang terputus")
        if not is_valid:
            messagebox.showerror("Papan Tidak Valid",
error_message)
        return

```

```

        analisis_warna = validasi_warna_papan(self.papan,
self.n)

        print(f"Hasil analisis warna papan: {analisis_warna}")

        self.file_label.config(
            text = f"
{os.path.basename(filepath)}\n{self.n}x{self.n} papan",
            fg = "green"
        )

        self._display_info(analisis_warna)
        self._draw_board(self.papan, queens=None)

        self.solve_btn.config(state=tk.NORMAL)

    except Exception as e:
        messagebox.showerror("Error", f"Gagal membaca
file:\n{str(e)}")

    def _display_info(self, analisis_warna):

        self.info_text.config(state = tk.NORMAL)
        self.info_text.delete(1.0, tk.END)

        info = f"Ukuran Papan: {self.n} x {self.n} \n"
        info += f"Warna Unik : {analisis_warna['jum_warna']}\n"
        info += f"Semua Warna:
{' '.join(sorted(analisis_warna['semua_warna']))}\n\n"
        self.info_text.insert(1.0, info)
        self.info_text.config(state=tk.DISABLED)

    def _update_speed(self, value):
        speed = float(value)

        if speed < 20:
            self.animation_speed = 0.001
            label = "Sangat Cepat"

        elif speed < 40:

```

```

        self.animation_speed = 0.01
        label = "Cepat"

    elif speed < 60:
        self.animation_speed = 0.05
        label = "Sedang"

    elif speed < 80:
        self.animation_speed = 0.1
        label = "Lambat"

    else:
        self.animation_speed = 0.2
        label = "Sangat Lambat"

    self.speed_value_label.config(text=label)

def _draw_board(self, papan, queens=None):
    self.canvas.delete("all")
    self.canvas_cells.clear()

    if not papan:
        return
    n = len(papan)

    max_ukuran_canva = 600
    self.cell_size = min(50, max_ukuran_canva // n)

    for row in range(n):
        for col in range(n):
            # Koordinat tiap satuan kotak
            x1 = col * self.cell_size
            y1 = row * self.cell_size
            x2 = x1 + self.cell_size
            y2 = y1 + self.cell_size

            char_warna = papan[row][col]
            fill_warna = self.COLORS.get(char_warna, 'white')

```

```

        cell_id = self.canvas.create_rectangle(
            x1, y1, x2, y2,
            fill = fill_warna,
            outline = 'black',
            width = 1
        )

        self.canvas.create_text(
            x1 + 5, y1 + 5,
            text = char_warna,
            font = ("Times New Roman", 6, "bold"),
            anchor = tk.NW,
            fill = 'black'
        )

        self.canvas_cells[(row, col)] = cell_id

    if queens:
        for row, col in queens:
            self._taruh_queen(row, col)

    self.canvas.config(scrollregion=self.canvas.bbox("all"))

    def _taruh_queen(self, row, col):
        x = col * self.cell_size + self.cell_size // 2
        y = row * self.cell_size + self.cell_size // 2

        self.canvas.create_text(
            x, y,
            text = "#",
            font = ("Arial", int(self.cell_size * 0.6)),
            tags = "queen"
        )

    def _update_board_visual(self, state_papan, posisi_queen):
        self._draw_board(self.papan, posisi_queen)

    def _start_solving(self):
        if self.is_solving:

```

```

        return

    self.is_solving = True
    self.result = None
    self.solve_btn.config(state=tk.DISABLED)
    self.select_file_btn.config(state=tk.DISABLED)

    self.status_label.config(text="Dalam proses penyelesaian")
    self.stats_text.config(state=tk.NORMAL)
    self.stats_text.delete(1.0, tk.END)
    self.stats_text.config(state=tk.DISABLED)

    self.solver_thread =
Thread(target=self._solver_worker, daemon=True )
    self.solver_thread.start()

    def _solver_worker(self):
        try:
            solver = SolverUtama(self.papan, self.n)
            def progress_callback(state_papan, kasus,
posisi_queen):
                if not self.is_solving:
                    return

                self.update_queue.put(('visual_update',
posisi_queen, kasus))
                time.sleep(self.animation_speed)

            result =
solver.solve_visual_callback(progress_callback)
            self.update_queue.put(('done', result))

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

    def _process_queue(self):
        try:
            while True:

```



```

        msg = self.update_queue.get_nowait()

        if msg[0]=='visual_update':
            _, posisi_queen, kasus = msg
            self._update_board_visual(None, posisi_queen)
            self._update_stats(kasus)

        elif msg[0] == 'done':
            _, result = msg
            self._finish_solving(result)

        elif msg[0] == 'error':
            _, error = msg
            self._handle_error(error)

    except queue.Empty:
        pass
    self.root.after(50, self._process_queue)

def _update_stats(self, kasus):
    self.stats_text.config(state=tk.NORMAL)
    self.stats_text.delete(1.0, tk.END)
    self.stats_text.insert(1.0, f"Eksplore Kasus:
{kasus:,}\n\n")
    self.stats_text.config(state=tk.DISABLED)

def _finish_solving(self, result: HasilSolusi):
    self.is_solving = False
    self.result = result

    self.solve_btn.config(state=tk.NORMAL)
    self.select_file_btn.config(state=tk.NORMAL)

    if result.found:
        self.status_label.config(text="Solusi ditemukan!")

    posisi_queen = []
    for row in range(self.n):
        for col in range(self.n):

```

```

        if result.solusi[row][col]=="#":
            posisi_queen.append((row,col))

    self._draw_board(self.papan, posisi_queen)

    self.stats_text.config(state=tk.NORMAL)
    self.stats_text.delete(1.0, tk.END)
    stats = f"SELESAI!\n\n"
    stats += f"Time: {result.waktu_eksekusi_ms:.2f} ms\n"
    stats += f"Cases: {result.jumlah_kasus:,}\n\n"
    stats += f"Queens placed: {self.n}"
    self.stats_text.insert(1.0, stats)
    self.stats_text.config(state=tk.DISABLED)

    self.save_btn.config(state=tk.NORMAL)
    messagebox.showinfo("SUSKES!!!", f"DITEMUKAN
SOLUSI!\n\nWaktu Pencarian: {result.waktu_eksekusi_ms:.3f} ms\n
Banyak Kasus yang Ditinjau {result.jumlah_kasus} kasus.")

    else:
        self.status_label.config(text="Tidak Ditemukan
Solusi")

        self.stats_text.config(state=tk.NORMAL)
        self.stats_text.delete(1.0, tk.END)
        stats = f"TIDAK ADA SOLUSI!\n"
        stats += f"Time: {result.waktu_eksekusi_ms:.2f} ms\n"
        stats += f"Cases: {result.jumlah_kasus:,}\n\n"
        stats += "Puzzle ini tidak dapat diselesaikan!\nTidak
ada solusi."
        self.stats_text.insert(1.0, stats)
        self.stats_text.config(state=tk.DISABLED)

        messagebox.showwarning("Tidak ada Solusi!!!", f"Tidak
ditemukan solusi\n\nWaktu Pencarian:
{result.waktu_eksekusi_ms:.3f} ms\n Banyak Kasus yang Ditinjau
{result.jumlah_kasus} kasus.")

    def _handle_error(self, error_msg: str):

```

```

self.is_solving=False

self.solve_btn.config(state=tk.NORMAL)
self.select_file_btn.config(state=tk.NORMAL)

self.status_label.config(text="Terjadi Error")
messagebox.showerror("Error", f"Solver
error:\n{error_msg}")

def _save_solution(self):
    if not self.result or not self.result.found:
        messagebox.showwarning("Peringatan", "Belum ada solusi
untuk disimpan!")
        return

    choice = messagebox.askquestion(
        "Format File",
        "Simpan sebagai:\n\nYes = Text (.txt)\nNo = Gambar
(.png)"
    )

    if choice == 'yes':
        filepath = filedialog.asksaveasfilename(
            title="Simpan Solusi",
            defaultextension=".txt",
            filetypes=[("Text Files", "*.txt"), ("All Files",
"*.*)"]
        )

        if not filepath:
            return

        try:
            from io_code import tulis_file_solusi
            tulis_file_solusi(
                filepath,
                self.result.solusi,
                self.result.waktu_eksekusi_ms,
                self.result.jumlah_kasus

```

```

        )
        messagebox.showinfo("Berhasil", f"Solusi disimpan
ke:\n{filepath}")
    except Exception as e:
        messagebox.showerror("Error", f"Gagal
menyimpan:\n{str(e)}")

    else:
        filepath = filedialog.asksaveasfilename(
            title="Simpan Gambar",
            defaultextension=".png",
            filetypes=[("PNG Image", "*.png"), ("JPEG Image",
            "*.jpg"), ("All Files", ".*")]
        )

        if not filepath:
            return

        try:
            self._save_canvas_screenshot(filepath)

            messagebox.showinfo("Berhasil", f"Gambar disimpan
ke:\n{filepath}")
        except Exception as e:
            messagebox.showerror("Error", f"Gagal menyimpan
gambar:\n{str(e)}")

    def _save_canvas_screenshot(self, filepath: str):
        n = self.n
        scale = 2
        cell = self.cell_size * scale
        img_w = n * cell
        img_h = n * cell

        img = Image.new("RGB", (img_w, img_h), "white")
        draw = ImageDraw.Draw(img)

        try:
            font_label = ImageFont.truetype("arialbd.ttf",

```

```

int(cell * 0.2))
        font_queen = ImageFont.truetype("arialbd.ttf",
int(cell * 0.7))
    except:
        font_label = ImageFont.load_default()
        font_queen = font_label

    queen_positions = {
        (int(self.canvas.coords(q)[1] // self.cell_size),
int(self.canvas.coords(q)[0] // self.cell_size))
        for q in self.canvas.find_withtag("queen")
    }

    for row in range(n):
        for col in range(n):
            x1, y1 = col * cell, row * cell
            x2, y2 = x1 + cell, y1 + cell

            fill = self.COLORS.get(self.papan[row][col],
"white")

            draw.rectangle([x1, y1, x2, y2], fill=fill,
outline="black")

            draw.text((x1 + 5, y1 + 5),
                    self.papan[row][col],
                    fill="black",
                    font=font_label)

            if (row, col) in queen_positions:
                draw.text((x1 + cell//2, y1 + cell//2),
                        "#",
                        fill="black",
                        font=font_queen,
                        anchor="mm")

    img.save(filepath)

def main():

```

```

    root = tk.Tk()
    app = QueenGUIVisual(root)
    root.mainloop()

if __name__ == "__main__":
    main()

```

2.2. io_code.py

```

import os
from typing import List, Tuple

def baca_file_papan(filepath: str) -> Tuple[List[List[str]], int]:
    if not os.path.exists(filepath):
        raise FileNotFoundError(f"File '{filepath}' tidak dapat ditemukan!")

    with open(filepath, 'r') as file:
        lines = file.readlines()
        lines = [line.strip() for line in lines if line.strip()]

    if not lines:
        raise ValueError(f"File tidak boleh kosong!")
    papan = []
    for line in lines:
        row = list(line)
        papan.append(row)
    n = len(papan)
    if n == 0:
        raise ValueError(f"Papan gak boleh kosong!")

    return papan, n

def tulis_file_solusi(filepath: str, solusi: List[List[str]],
waktu_eksekusi_ms: float, jumlah_kasus: int):
    with open(filepath, 'w') as file:
        for row in solusi:

```

```

        file.write(''.join(row) + '\n')

        file.write(f'\nWaktu pencarian: {waktu_eksekusi_ms:.2f}
ms\n')

        file.write(f'Banyak kasus yang ditinjau: {jumlah_kasus}
kasus\n')

def papan_ke_str(papan: List[List[str]]) -> str:
    return '\n'.join(''.join(row) for row in papan)

def txt_in_folder(folder: str) -> List[str]:
    if not os.path.exists(folder):
        return []
    return [f for f in os.listdir(folder) if f.endswith('.txt')]

def cek_folder(folder: str):
    if not os.path.exists(folder):
        os.makedirs(folder)

```

2.3. solver_core.py

```

from typing import List, Dict, Tuple, Optional, Callable
from dataclasses import dataclass
import time

@dataclass
class HasilSolusi:
    solusi : Optional[List[List[str]]]
    jumlah_kasus : int
    waktu_eksekusi_ms: float
    found: bool

class SolverUtama:

```

```

def __init__(self, papan: List[List[str]], n: int):
    self.papan_awal = papan
    self.n = n
    self.jumlah_kasus = 0

    self.lokasi_warna = self._bangun_lokasi_warna()

    self.warna_ada_queen = set()

    self.papan_solusi = [row[:] for row in papan]

def _semua_kombinasi(self, panjang: int) -> List[List[int]]:
    if panjang == 0:
        return [[]]
    hasil = []
    for sub in self._semua_kombinasi(panjang - 1):
        for col in range(self.n):
            hasil.append(sub + [col])
    return hasil

def _bangun_lokasi_warna(self) -> Dict[str, List[Tuple[int,
int]]]:
    map_warna = {}
    for row in range(self.n):
        for col in range(self.n):
            warna = self.papan_awal[row][col]
            if warna not in map_warna:
                map_warna[warna] = []
            map_warna[warna].append((row, col))

    return map_warna

def _bisa_lanjut(self, row: int, col: int, posisi_queenn:
List[Tuple[int, int]]) -> bool:
    warna = self.papan_awal[row][col]
    for rowQ, colQ in posisi_queenn:
        if rowQ == row or colQ == col:

```



```

        return False
    if (abs(rowQ - row) <= 1) and (abs(colQ - col) <= 1):
        return False
    if (self.papan_awal[rowQ][colQ] == warna):
        return False
    return True

def _cek_constraint(self, permutasi) -> bool:
    posisi = []
    for row, col in enumerate(permutasi):
        if not self._bisa_lanjut(row, col, posisi):
            return False
        posisi.append((row, col))
    return True

def _backtrack(self, row: int, posisi_queen: List[Tuple[int,
int]], progress_callback: Optional[Callable] = None) -> bool:
    if row == self.n:
        return True

    for col in range(self.n):
        self.jumlah_kasus += 1

        if self._bisa_lanjut(row, col, posisi_queen):
            warna = self.papan_awal[row][col]
            self.warna_ada_queen.add(warna)
            posisi_queen.append((row, col))
            self.papan_solusi[row][col] = '#'

            if progress_callback and self.jumlah_kasus % 100
== 0:
                tampilin_progress_papan = self._papan_ke_str()

            if self._backtrack(row+1, posisi_queen,
progress_callback):
                return True

            posisi_queen.pop()
            self.warna_ada_queen.remove(warna)

```

```

        self.papan_solusi[row][col] =
self.papan_awal[row][col]
        return False

    def _papan_ke_str(self) -> str:
        return '\n'.join(''.join(row) for row in
self.papan_solusi)

    def _bangun_papan_solusi(self, permutasi):
        self.papan_solusi = [row[:] for row in self.papan_awal]
        for row, col, in enumerate(permutasi):
            self.papan_solusi[row][col] = "#"

    def solve(self, progress_callback: Optional[Callable[[str,
int], None]] = None) -> HasilSolusi:
        waktu_mulai = time.time()
        self.jumlah_kasus = 0
        posisi_queen = []
        result = self._backtrack(0, posisi_queen,
progress_callback)

        waktu_eksekusi = (time.time() - waktu_mulai) * 1000
        if result:
            return HasilSolusi(
                solusi = self.papan_solusi,
                jumlah_kasus = self.jumlah_kasus,
                waktu_eksekusi_ms = waktu_eksekusi,
                found = True,
            )
        else:
            return HasilSolusi(
                solusi = None,
                jumlah_kasus = self.jumlah_kasus,
                waktu_eksekusi_ms = waktu_eksekusi,
                found = False,
            )

    def solve_visual_callback(self, visual_callback:

```

```

Optional[Callable[[str, int, List[Tuple[int, int]]], None]] =
None) -> HasilSolusi:
    waktu_mulai = time.time()
    self.jumlah_kasus = 0

    for kombinasi in self._semua_kombinasi((self.n)):
        self.jumlah_kasus+=1
        posisi_queen = [(row, kombinasi[row]) for row in
range(self.n)]
        if visual_callback:
            self._bangun_papan_soluso(kombinasi)
            gambar_papan = self._papan_ke_str()
            visual_callback(gambar_papan, self.jumlah_kasus,
posisi_queen)

        if self._cek_constraint(kombinasi):
            self._bangun_papan_soluso(kombinasi)
            waktu_eksekusi_ms = (time.time()-waktu_mulai)*1000
            return HasilSolusi(
                solusi = self.papan_solusi,
                jumlah_kasus = self.jumlah_kasus,
                waktu_eksekusi_ms = waktu_eksekusi_ms,
                found = True
            )
    waktu_eksekusi_ms = (time.time() - waktu_mulai)*1000
    return HasilSolusi(
        solusi = None,
        jumlah_kasus = self.jumlah_kasus,
        waktu_eksekusi_ms = waktu_eksekusi_ms,
        found = False
    )

    def _backtrack_visual(self, row: int, posisi_queen:
List[Tuple[int, int]], visual_callback: Optional[Callable] = None)
-> bool:
        if row == self.n:
            return True

        for col in range(self.n):

```

```

        self.jumlah_kasus += 1

        if self._bisa_lanjut(row, col, posisi_queen):
            warna = self.papan_awal[row][col]
            posisi_queen.append((row, col))
            self.warna_ada_queen.add(warna)
            self.papan_solusi[row][col] = '#'

            if visual_callback:
                board_snapshot = self._papan_ke_str()
                visual_callback(board_snapshot,
self.jumlah_kasus, posisi_queen[:])

                if self._backtrack_visual(row+1, posisi_queen,
visual_callback):

                    return True

            posisi_queen.pop()
            self.warna_ada_queen.remove(warna)
            self.papan_solusi[row][col] =
self.papan_awal[row][col]

            if visual_callback:
                board_snapshot = self._papan_ke_str()
                visual_callback(board_snapshot,
self.jumlah_kasus, posisi_queen[:])

        return False

def validasi_ukuran_papan(papan: List[List[str]], n: int) ->
Tuple[bool, str]:
    for i, row in enumerate(papan):
        if len(row) != n:
            return False, f"Row {i+1} punya panjang {len(row)},
seharusnya {n}"

    for row in range(n):

```

```

        for col in range(n):
            if not papan[row][col].isalpha() or not
papan[row][col].isupper():
                return False, f"Papan tidak sesuai di posisi
({row}, {col}), hanya menerima huruf kapital A-Z!"

        return True, ""

def validasi_warna_papan(papan: List[List[str]], n: int) ->
Dict[str, any]:
    semua_warna = set()
    map_jum_warna = {}

    for row in papan:
        for warna in row:
            semua_warna.add(warna)
            if warna not in map_jum_warna:
                map_jum_warna[warna] = 1
            else:
                map_jum_warna[warna] += 1

    jum_warna = len(semua_warna)

    mungkin_ada_solusi = (jum_warna >= n)
    warning = ""
    if jum_warna < n:
        warning = (f"Papan cuman punya {jum_warna} warna tapi ada
{n} queens. ")
        f"Tiap warna cuman bisa 1 queen, Tidak ada
solusi")
    elif jum_warna > n:
        warning = (f"Papan punya {jum_warna} warna untuk {n}
queens. ")
        f"{jum_warna - n} warna akan kosong. Kemungkinan
aman, bisa ada solusi.")

    return {
        'jum_warna': jum_warna,
        'semua_warna': semua_warna,

```

```

        'map_jum_warna' : map_jum_warna,
        'mungkin_ada_solusi': mungkin_ada_solusi,
        'pesan_warning': warning
    }

def cek_konek_warna(papan: List[List[str]]) -> Tuple[bool,
List[str]]:
    n = len(papan)
    map_warna: Dict[str, List[Tuple[int, int]]] = {}
    for row in range(n):
        for col in range(n):
            warna = papan[row][col]
            if warna not in map_warna:
                map_warna[warna] = []
            map_warna[warna].append((row, col))
    warna_terpisah = []

    for warna, koordinats in map_warna.items():
        if len(koordinats) == 1:
            continue
        sel_unik = set(koordinats)
        visited = set()
        antrian = [koordinats[0]]
        visited.add(koordinats[0])

        while antrian:
            rowA, colA = antrian.pop(0)
            for r, c in [(-1,0), (1,0), (0,-1), (0,1)]:
                tetangga = (r+rowA, c+colA)
                if tetangga in sel_unik and tetangga not in
visited:
                    visited.add(tetangga)
                    antrian.append(tetangga)

            if len(visited) != len(koordinats):
                warna_terpisah.append(warna)

    valid = len(warna_terpisah) == 0
    return valid, warna_terpisah

```

BAB 3

TEST CASE

3.1. Test_4x4.txt

- Input:

AABB
AABB
CCDD
CCDD

- Output:

A	A #	B	B
A	A	B	B #
C #	C	D	D
C	C	D #	D

3.2. Test_4x4_v2.txt

- Input:

AABB
ACCB
ACDD
ACDD

- Output:

A	A	B #	B
A #	C	C	B
A	C	D	D #
A	C #	D	D

3.3. Test_5x5.txt

- Input:

```
AAABB
ACCCB
ACDBB
EEDDB
EEEDB
```

- Output:

A #	A	A	B	B
A	C	C #	C	B
A	C	D	B	B #
E	E #	D	D	B
E	E	E	D #	B

3.4. Test_6x6.txt

- Input:

```
AAAABB
CCADDB
CEEEDB
```



```
CEEFDDB
CCFFFB
CCCFFF
```

- Output:

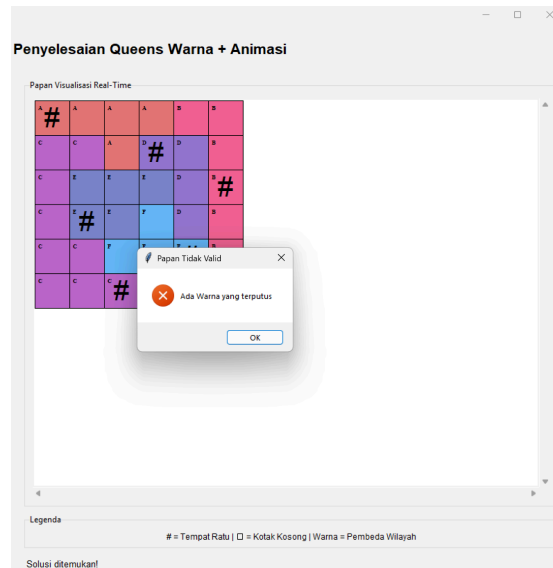
A #	A	A	A	B	B
C	C	A	D #	D	B
C	E	E	E	D	B #
C	E #	E	F	D	B
C	C	F	F	F #	B
C	C	C #	F	F	F

3.5. Test_6x6_konekProblem.txt

- Input:

```
ABBA
BBCC
DCCC
DADA
```

- Output:

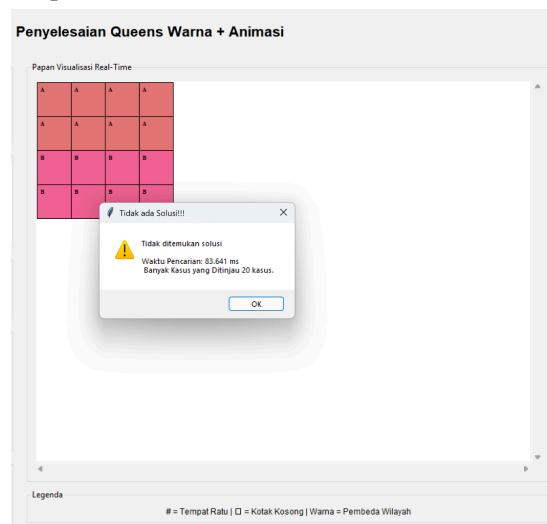


3.6. Test_Impossible.txt

- Input:

```
AAAA
AAAA
BBBB
BBBB
```

- Output:



BAB 4

LAMPIRAN

Source code lengkap program tersedia pada repository GitHub berikut ini:

Tucil1_13524079 : Penyelesaian Queens LinkedIn dengan Algoritma Brute Force

https://github.com/andraalanna/Tucil1_13524079

BAB 5

REFERENSI

Berikut adalah referensi yang digunakan dalam pengembangan program untuk memenuhi Tugasil Strategi Algoritma

1. Dokumentasi Resmi

[1] Python Software Foundation. tkinter - Python interface to Tcl/Tk. Python 3 Documentation. Diakses dari: <https://docs.python.org/3/library/tkinter.html>

Digunakan sebagai referensi utama dalam pembuatan GUI program, mencakup penggunaan widget dasar seperti Label, Button, Frame, Text, Scale, dan Canvas.

[2] Python Software Foundation. Pillow (PIL Fork) Documentation - Image Module.

Diakses dari: <https://pillow.readthedocs.io/en/stable/reference/Image.html>

Digunakan sebagai referensi untuk membuat dan menyimpan gambar solusi. Modul Image dan ImageDraw dari library Pillow dipakai untuk menggambar kotak warna dan simbol queen, lalu menyimpannya sebagai file .png.

[3] Python Software Foundation. Pillow (PIL Fork) Documentation - ImageGrab Module.

Diakses dari: <https://pillow.readthedocs.io/en/stable/reference/ImageGrab.html>

Digunakan sebagai referensi untuk fitur save canvas sebagai gambar menggunakan ImageGrab.grab(), yang melakukan screenshot pada area canvas tkinter dan menyimpannya sebagai file gambar.

2. Video Tutorial

[4] Tkinter and Threads - How to Prevent Your GUI From Freezing. YouTube.

Diakses dari: <https://www.youtube.com/watch?v=jnrCpA1xJPQ&t=302s>

Digunakan sebagai referensi dalam mengimplementasikan threading pada GUI. Video ini menjelaskan cara menjalankan proses berat (seperti solver) di background thread agar GUI tidak freeze selama proses solving berlangsung.

[5] Python Queue Tutorial. YouTube.

Diakses dari: https://www.youtube.com/watch?v=6zmI_BU18xk

Digunakan sebagai referensi dalam penggunaan queue.Queue() untuk komunikasi antara solver thread dan main GUI thread secara thread-safe.

[6] Python Queue - Multithreading. YouTube.

Diakses dari: https://www.youtube.com/watch?v=SO_Yc5bydgl

Digunakan sebagai referensi tambahan mengenai implementasi queue dalam konteks multithreading Python, khususnya untuk mengirimkan update posisi queen dari worker thread ke GUI thread.

[7] Tkinter filedialog - Open and Save Files. YouTube.

Diakses dari: https://youtu.be/Aim_7fC-inw?si=fCgBaD10pJVD6Kbg

Digunakan sebagai referensi dalam mengimplementasikan `filedialog.askopenfilename()` untuk memilih file input `.txt` dan `filedialog.asksaveasfilename()` untuk menyimpan solusi ke file output.

[8] Python Tkinter Canvas Tutorial. YouTube.

Diakses dari: https://youtu.be/HrK9Kmz3_9A

Digunakan sebagai referensi dalam mengimplementasikan Canvas widget untuk menampilkan visualisasi board secara real-time. Canvas dipakai untuk menggambar kotak-kotak berwarna dan simbol queen (#) yang bergerak selama proses animasi backtracking berlangsung.

BAB 6

LAMPIRAN

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	✓	

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

A stylized, handwritten signature in black ink, consisting of a large, bold letter 'A' with a horizontal crossbar and a vertical stem, followed by a smaller, more fluid script.

Angelina Andra Alanna