LAPORAN TUGAS KECIL 1 MATA KULIAH IF2211 STRATEGI ALGORITMA

Penyelesaian Permainan Queens LinkedIn Menggunakan Algoritma Brute Force

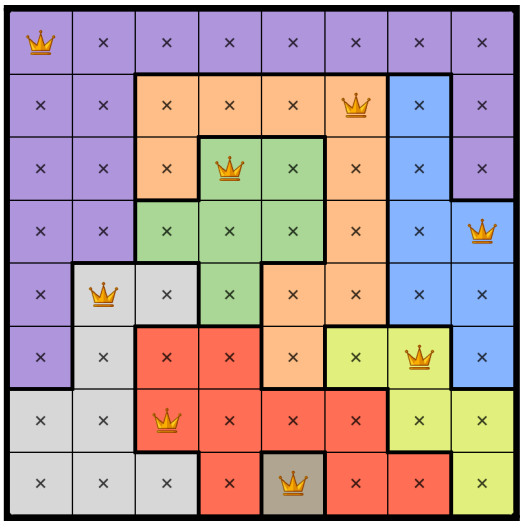
Dosen Pengampu :
Prof. Dr. Ir. Rinaldi, M.T.

Disusun Oleh:
Matthew Sebastian Kurniawan - 18223096
Kelas - K01

Program Studi Sistem dan Teknologi Informasi Sekolah Teknik Elektro dan Informatika- Institut Teknologi Bandung Jl. Ganesha 10, Bandung 4013

DAFTAR ISI

BAB I
PERMASALAHAN



Permainan Queens LinkedIn merupakan permainan papan berbasis logika yang mengharuskan pemain untuk menempatkan sejumlah queen pada sebuah papan persegi yang terdiri atas beberapa daerah berwarna. Tujuan dari permainan ini adalah menentukan posisi queen sedemikian rupa sehingga seluruh aturan permainan terpenuhi secara bersamaan. Setiap queen yang ditempatkan akan memengaruhi kemungkinan penempatan queen lainnya, karena terdapat berbagai batasan yang harus dipatuhi. Tantangan utama dalam permainan ini terletak pada bagaimana menemukan konfigurasi penempatan queen yang valid di antara banyak kemungkinan susunan yang dapat terbentuk.

Dalam permainan ini, terdapat beberapa aturan utama yang harus dipenuhi sebagai berikut.

- Setiap baris hanya boleh memiliki tepat satu queen.
- Setiap kolom hanya boleh memiliki tepat satu queen.
- Setiap daerah warna hanya boleh memiliki tepat satu queen.
- Tidak ada dua queen yang saling bersebelahan, baik secara horizontal, vertikal, maupun diagonal.

Permasalahan muncul karena setiap penempatan queen harus mempertimbangkan seluruh aturan tersebut secara bersamaan. Satu kesalahan kecil dalam penempatan dapat menyebabkan konflik pada baris, kolom, daerah warna, maupun posisi yang bersebelahan. Oleh karena itu, diperlukan pendekatan yang sistematis untuk menelusuri berbagai kemungkinan konfigurasi hingga diperoleh solusi yang memenuhi seluruh ketentuan permainan.

BAB II
PENDEKATAN SOLUSI

Pendekatan yang digunakan dalam program ini adalah algoritma brute force murni dengan melakukan eksplorasi seluruh kemungkinan penempatan n queen pada papan berukuran n x n. Program terlebih dahulu mengumpulkan seluruh posisi sel pada papan, kemudian menghasilkan semua kombinasi pemilihan n posisi dari total n^2 sel yang tersedia. Setiap kombinasi yang terbentuk dianggap sebagai satu konfigurasi kandidat solusi yang akan diperiksa lebih lanjut. Pendekatan ini bersifat exhaustive search karena seluruh kemungkinan konfigurasi dibuat tanpa menggunakan heuristik atau pembatasan awal.

Setelah kombinasi terbentuk, setiap konfigurasi diperiksa apakah memenuhi aturan permainan, yaitu tepat satu queen pada setiap baris, kolom, dan daerah warna, serta tidak ada dua queen yang saling bersebelahan termasuk secara diagonal. Program menyediakan dua mekanisme pencarian, yaitu metode yang men-*generate* seluruh kombinasi terlebih dahulu lalu memvalidasi semuanya, serta metode yang langsung memvalidasi saat kombinasi lengkap terbentuk dan menghentikan pencarian ketika solusi pertama ditemukan. Meskipun metode kedua dapat lebih cepat pada beberapa kasus, keduanya tetap memiliki kompleksitas kombinatorial karena ruang solusi yang diperiksa bersifat menyeluruh.

IMPLEMENTASI DALAM KODE

A) Konstruktor dan Inisialisasi

```
class LinkedinQueenSolver:
    def __init__(self):
        self.board = None
        self.n = 0
        self.solution = None
        self.total_possibilities = 0
        self.total_checked = 0
        self.time_ms = 0
        self.error_message = None
```

Struktur data utama disimpan dalam kelas LinkedinQueenSolver. Atribut board menyimpan representasi papan dalam bentuk matriks dua dimensi, sedangkan n menyimpan ukuran papan. Variabel solution digunakan untuk menyimpan konfigurasi queen yang valid apabila ditemukan. Selain itu, program juga menyimpan informasi jumlah kemungkinan total, jumlah konfigurasi yang diperiksa, serta waktu eksekusi dalam milidetik. Variabel error_message digunakan untuk menangani kesalahan input.

B) Membaca dan Memvalidasi Papan dari File

```
    def read_board(self, filename):
        self.error_message = None
        board = []

        with open(filename, "r") as file:
            for row in file:
                row = row.strip()
                if row:
                    board.append(list(row))

        n = len(board)
        for row in board:
            if len(row) != n:
                self.error_message = "input tidak valid, ukuran board bukan n x n"
                return None

        colors = set()
        for row in board:
            for col in row:
                colors.add(col)

        if len(colors) != n:
            self.error_message = "input tidak valid, warna melebihi jumlah n"
            return None

        self.board = board
        self.n = n
        return board
```

Fungsi read_board digunakan untuk membaca file papan permainan dan memastikan formatnya valid. Program mengecek apakah papan berbentuk matriks n×n dan memiliki tepat n warna unik. Jika tidak sesuai, pesan error disimpan dan program dihentikan. Jika valid, papan dan ukurannya disimpan untuk digunakan dalam proses pencarian solusi.

C) Validasi Konfigurasi Queen

```
    def is_valid(self, q_positions):
        n = len(q_positions)

        # Cek baris unik
        rows = [pos[0] for pos in q_positions]
        if len(set(rows)) != n:
            return False
        # Cek kolom unik
        cols = [pos[1] for pos in q_positions]
        if len(set(cols)) != n:
            return False
        # Cek warna unik
        used_colors = set()
        for row, col in q_positions:
            color = self.board[row][col]
            if color in used_colors:
                return False
            used_colors.add(color)
        # Cek tidak ada queen yang bersebelahan
        for i in range(n):
            for j in range(i + 1, n):
                row_i, col_i = q_positions[i]
                row_j, col_j = q_positions[j]
                if abs(row_i - row_j) <= 1 and abs(col_i - col_j) <= 1:
                    return False
        return True
```

Fungsi is_valid digunakan untuk memeriksa apakah konfigurasi queen memenuhi seluruh aturan permainan. Fungsi akan langsung mengembalikan False jika ditemukan pelanggaran, dan True jika seluruh aturan terpenuhi.

Pengecekan yang dilakukan adalah:

- Baris unik, memastikan setiap queen berada pada baris yang berbeda.
- Kolom unik, memastikan setiap queen berada pada kolom yang berbeda.
- Warna unik, memastikan tidak ada dua queen pada daerah warna yang sama.
- Tidak bersebelahan, memastikan tidak ada dua queen yang saling bersentuhan

secara horizontal, vertikal, maupun diagonal.

Jika semua kondisi terpenuhi, konfigurasi dinyatakan valid.

D) Generate Semua Kemungkinan

```
# Generate semua kemungkinan
def generate_all(self, queen_idx, q_positions, all_positions, start_idx,
all_possibility):
        if queen_idx == self.n:
            all_possibility.append(q_positions.copy())
            return

        for i in range(start_idx, len(all_positions)):
            pos = all_positions[i]
            q_positions.append(pos)
            self.generate_all(queen_idx + 1, q_positions, all_positions, i + 1,
all_possibility)
            q_positions.pop()
```

Fungsi generate_all digunakan untuk membuat seluruh kombinasi peletakan queen sebanyak n dari seluruh posisi papan. Proses dilakukan secara rekursif dengan menambahkan satu posisi queen pada setiap langkah hingga jumlah queen mencapai n. Setiap kombinasi lengkap disimpan dalam daftar all_possibility. Proses ini memastikan seluruh kemungkinan konfigurasi dibuat secara menyeluruh.

E) Memvalidasi Seluruh Kombinasi

```
    # Validasi
    def validate_all(self, all_configs):
        valid_solutions = []
        iteration = 0

        for config in all_configs:
            iteration += 1

            if iteration % 100 == 0:
                print(f"==== Iterasi ke-{iteration} ====")
```

```
                self.print_board_with_queens(config)

            if self.is_valid(config):
                valid_solutions.append(config)

        return valid_solutions
```

Fungsi validate_all digunakan untuk memeriksa seluruh konfigurasi queen yang telah dihasilkan sebelumnya. Setiap konfigurasi diuji menggunakan fungsi is_valid, dan jika memenuhi semua aturan maka dimasukkan ke dalam daftar solusi valid. Fungsi ini juga menampilkan papan setiap 100 iterasi untuk menunjukkan progres pencarian.

F) Generate dan Validasi Secara Langsung

```
    # Generate dan Cek
    def generate_and_check(self, queen_idx, q_positions, all_positions, start_idx,
result):
        if queen_idx == self.n:
            result['total'] += 1

            if result['total'] % 500 == 0:
                print(f"==== Iterasi ke-{result['total']} ====")
                self.print_board_with_queens(q_positions)

            if self.is_valid(q_positions):
                result['solution'] = q_positions.copy()
                return True

            return False

        for i in range(start_idx, len(all_positions)):
            pos = all_positions[i]
            q_positions.append(pos)

            if self.generate_and_check(queen_idx + 1, q_positions, all_positions, i
+ 1, result):
                return True
```

```
            q_positions.pop()

        return False
```

Fungsi generate_and_check digunakan untuk menghasilkan kombinasi queen sekaligus langsung memvalidasinya tanpa menyimpan semua kemungkinan terlebih dahulu. Jika suatu konfigurasi valid ditemukan, proses pencarian langsung dihentikan sehingga lebih efisien dibanding brute force biasa.

G) Menampilkan Papan dengan Posisi Queen

```
    def print_board_with_queens(self, q_positions):
        queen_positions = set(q_positions)
        for row in range(self.n):
            line_output = ""
            for col in range(self.n):
                if (row, col) in queen_positions:
                    line_output += "# "
                else:
                    line_output += self.board[row][col] + " "
            print(line_output)
```

Fungsi print_board_with_queens digunakan untuk menampilkan papan permainan beserta posisi queen dalam suatu konfigurasi tertentu. Posisi queen ditandai dengan simbol #, sedangkan kotak lainnya menampilkan warna asli papan.

H) Brute Force (Generate Semua – Validasi Semua)

```
    # brute force
    def brute_force(self):
        all_positions = []
        for r in range(self.n):
            for c in range(self.n):
                all_positions.append((r, c))

        start_time = time.time()
        all_possibility = []
```

```
        self.generate_all(0, [], all_positions, 0, all_possibility)
        self.total_possibilities = len(all_possibility)
        self.total_checked = len(all_possibility)
        valid_solutions = self.validate_all(all_possibility)
        self.solution = valid_solutions[0] if valid_solutions else None
        end_time = time.time()
        self.time_ms = (end_time - start_time) * 1000

        return self.solution, self.total_checked, self.time_ms
```

Fungsi brute_force digunakan untuk menjalankan metode brute force penuh, yaitu dengan menghasilkan seluruh kemungkinan kombinasi peletakan queen terlebih dahulu, kemudian memvalidasi semuanya satu per satu. Metode ini sederhana namun kurang efisien karena tetap memeriksa semua kemungkinan meskipun solusi sudah ditemukan.

I) Optimized Brute Force (Generate dan Validasi Secara Langsung)

```
    # Optimized brute force
    def optimized_brute_force(self):
        all_positions = []
        for r in range(self.n):
            for c in range(self.n):
                all_positions.append((r, c))

        all_possibility = []
        self.generate_all(0, [], all_positions, 0, all_possibility)
        self.total_possibilities = len(all_possibility)

        start_time = time.time()

        result = {'solution': None, 'total': 0}

        self.generate_and_check(0, [], all_positions, 0, result)

        self.solution = result['solution']
        self.total_checked = result['total']
```

```
        end_time = time.time()
        self.time_ms = (end_time - start_time) * 1000

        return self.solution, self.total_checked, self.time_ms
```

Fungsi optimized_brute_force digunakan untuk menjalankan versi optimasi dari brute force, yaitu dengan melakukan generate dan validasi secara langsung. Metode ini tetap menghitung total kemungkinan, tetapi dapat berhenti lebih cepat saat solusi ditemukan sehingga waktu eksekusi lebih singkat.

J) Menampilkan Solusi Akhir

```
    def print_solution(self):
        if self.solution is None:
            print("Tidak ada solusi.")
            return

        queen_positions = set(self.solution)

        for row in range(self.n):
            line_output = ""
            for col in range(self.n):
                if (row, col) in queen_positions:
                    line_output += "#"
                else:
                    line_output += self.board[row][col]
            print(line_output)
```

Fungsi print_solution digunakan untuk menampilkan solusi akhir yang ditemukan dalam bentuk papan. Jika tidak ada solusi, fungsi akan menampilkan pesan bahwa solusi tidak tersedia.

K) Program Utama

```
if __name__ == "__main__":
    solver = LinkedinQueenSolver()
    filename = input("Masukkan nama file (.txt): ")
```

```
    if not os.path.exists(filename):
        test_path = os.path.join(os.path.dirname(__file__), "..", "test", filename)
        if os.path.exists(test_path):
            filename = test_path
    result = solver.read_board(filename)

    if result is None:
        print(solver.error_message)
        exit()
    print("\nPilih metode:")
    print("1. Brute Force (generate semua - validasi semua)")
    print("2. Optimized Brute Force (generate 1 - validasi - stop kalau ketemu)")
    pilihan = input("Masukkan pilihan (1/2): ")

    if pilihan == "2":
        print("\nMenggunakan Optimized Brute Force")
        solution, total_checked, time_ms = solver.optimized_brute_force()
    else:
        print("\nMenggunakan Brute Force")
        solution, total_checked, time_ms = solver.brute_force()
    if solution:
        print("\nSolusi ditemukan:\n")
        solver.print_solution()
    else:
        print("\nTidak ada solusi.")
    print(f"\nUkuran papan: {solver.n} x {solver.n}")
    print(f"Waktu pencarian: {round(time_ms, 2)} ms")
    print(f"Banyak kemungkinan: {solver.total_possibilities} kasus")
    print(f"Banyak kasus yang ditinjau: {solver.total_checked} kasus")
```

Bagian utama program digunakan untuk menjalankan keseluruhan alur aplikasi, mulai dari membaca file papan, memvalidasi input, meminta pengguna memilih metode pencarian, menjalankan algoritma, hingga menampilkan hasil berupa solusi, waktu pencarian, jumlah kemungkinan, dan jumlah kasus yang ditinjau.

BAB III
ANALISIS KOMPLEKSITAS

A) Brute Force (Generate Semua – Validasi Semua)

Pada method brute_force(), algoritma pertama-tama membentuk seluruh daftar posisi pada papan berukuran n × n sebanyak n^2 sel. Kemudian fungsi generate_all() menghasilkan seluruh kemungkinan kombinasi pemilihan n posisi dari total n^2 posisi. Jumlah kombinasi yang dihasilkan adalah:

$$C(n^2, n)$$

Setelah seluruh konfigurasi dihasilkan dan disimpan dalam all_possibility, method validate_all() akan memeriksa setiap konfigurasi menggunakan is_valid(). Proses validasi ini memiliki kompleksitas $O(n^2)$ pada skenario terburuk karena terdapat pemeriksaan pasangan queen (nested loop).

Maka, kompleksitas waktu method brute_force() adalah:

$$O(C(n^2, n) \times n^2)$$

Karena pertumbuhan kombinasi $C(n^2,n)$ mendekati orde n^{2n}, maka setelah disederhanakan secara asimtotik kompleksitasnya dapat dinyatakan sebagai:

$$O(n^{2n})$$

B) Optimized Brute Force (Generate dan Validasi Secara Langsung)

Pada method optimized_brute_force(), proses pembuatan konfigurasi dan validasi dilakukan secara bersamaan melalui fungsi rekursif generate_and_check(). Berbeda dengan brute_force() yang selalu menghasilkan seluruh kemungkinan konfigurasi terlebih dahulu, pendekatan ini akan langsung berhenti ketika solusi pertama ditemukan.

Dalam kondisi best case, konfigurasi pertama yang terbentuk sudah memenuhi seluruh constraint (baris unik, kolom unik, warna unik, dan tidak bersebelahan). Akibatnya, algoritma hanya melakukan satu kali pembentukan konfigurasi lengkap dan satu kali pemanggilan fungsi is_valid().

Karena fungsi is_valid() memiliki kompleksitas waktu $O(n^2)$ pada skenario terburuk (akibat pemeriksaan pasangan queen), maka pada best case total kompleksitas waktunya menjadi:

$$\boxed{O(n^2)}$$

Dengan demikian, kompleksitas best case untuk optimized_brute_force() adalah $O(n^2)$, sedangkan kompleksitas worst case tetap $O(C(n^2,n) \times n^2)$ yang setelah disederhanakan menjadi $O(n^{2n})$. Perbedaan ini terjadi karena adanya mekanisme penghentian lebih awal yang tidak dimiliki oleh metode brute force biasa.

BAB IV
BONUS

1) Membuat GUI (Graphical User Interface)

Program ini, menggunakan modul tkinter untuk membangun GUI utama LinkedIn Queen Solver, yang memungkinkan pengguna untuk memasukkan papan awal dari file atau gambar, menampilkan papan pada canvas, menampilkan statistik seperti ukuran papan, waktu eksekusi, total kemungkinan konfigurasi dan total konfigurasi yang diperiksa, serta men-trigger algoritma penyelesaian. Kelas utama adalah LinkedinQueenGUI(root) yang menerima parameter root sebagai Tkinter window, di dalamnya terdapat fungsi setup_ui() untuk membuat layout frame, canvas, tombol, dan entry board size, draw_board() untuk menggambar papan beserta posisi queen jika ada solusi, dan update_info() untuk memperbarui label statistik sesuai kondisi solver.

IMPLEMENTASI DALAM KODE

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
from LinkedinQueenSolver import LinkedinQueenSolver

class LinkedinQueenGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("LinkedIn Queen Solver")
        self.root.configure(bg='#1a1a2e')
        self.root.resizable(False, False)

        self.solver = LinkedinQueenSolver()
        self.color_map = {}
```

```
        self.color_palette = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4',
'#FFEAA7']
        self.board_size = 4

        self.setup_ui()

    def setup_ui(self):
        bg_color = '#1a1a2e'
        text_color = '#eaeaea'
        button_color = '#0f3460'
        self.main_frame = tk.Frame(self.root, bg=bg_color, padx=20, pady=20)
        self.main_frame.pack()

        self.lbl_title = tk.Label(self.main_frame, text="LinkedIn Queen Solver",
                                  font=("Arial", 24, "bold"), bg=bg_color,
fg='#e94560')
        self.lbl_title.pack(pady=(0, 20))

        self.canvas_size = 450
        self.canvas = tk.Canvas(self.main_frame, width=self.canvas_size,
height=self.canvas_size,
                                bg='#16213e', highlightthickness=3,
highlightbackground='#e94560')
        self.canvas.pack()

    def draw_board(self):
        self.canvas.delete("all")
        if self.solver.board is None:
            return
        cell_size = self.canvas_size // self.solver.n
        offset = (self.canvas_size - cell_size * self.solver.n) // 2
        q_positions = set(self.solver.solution) if self.solver.solution else set()
        for r in range(self.solver.n):
            for c in range(self.solver.n):
                x1 = offset + c * cell_size
                y1 = offset + r * cell_size
```

```
                x2 = x1 + cell_size
                y2 = y1 + cell_size
                char = self.solver.board[r][c]
                color = self.color_map.get(char, 'lightgray')
                self.canvas.create_rectangle(x1, y1, x2, y2, fill=color,
outline='#1a1a2e', width=2)
                if (r, c) in q_positions:
                    padding = cell_size // 6
                    self.canvas.create_oval(x1+padding, y1+padding, x2-padding,
y2-padding,
                                            fill='white', outline='#1a1a2e',
width=2)
                    font_size = max(12, cell_size // 2)
                    self.canvas.create_text((x1+x2)//2, (y1+y2)//2, text="Q",
font=("Arial", font_size), fill='#1a1a2e')

    def update_info(self):
        if self.solver.board:
            self.lbl_board_size_info.config(text=f"Board Size: {self.solver.n} x
{self.solver.n}")
        else:
            self.lbl_board_size_info.config(text="Board Size: -")

        if self.solver.time_ms > 0:
            self.lbl_time.config(text=f"Time: {self.solver.time_ms:.2f} ms")
        else:
            self.lbl_time.config(text="Time: -")

        if self.solver.total_possibilities > 0:
            self.lbl_total_possible.config(text=f"Total Possible\nConfigurations:
{self.solver.total_possibilities}")
        else:
            self.lbl_total_possible.config(text="Total Possible\nConfigurations:
-")

        if self.solver.total_checked > 0:
```

```
            self.lbl_total_checked.config(text=f"Total Configurations\nChecked:
{self.solver.total_checked}")
        else:
            self.lbl_total_checked.config(text="Total Configurations\nChecked: -")

if __name__ == "__main__":
    root = tk.Tk()
    app = LinkedinQueenGUI(root)
    root.mainloop()
```

2) Input dari Image

Program ini, menggunakan modul tkinter dan PIL untuk memungkinkan input papan awal berupa gambar, dimana kelas LinkedinQueenGUI memiliki fungsi load_image() yang membuka dialog file image, display_image_on_canvas() yang menampilkan image pada canvas, dan extract_colors_from_image() yang membagi image menjadi sel berdasarkan ukuran papan (board_size) lalu mengekstrak warna dominan tiap sel, kemudian memetakan warna menjadi karakter unik untuk dijadikan board awal, fungsi get_dominant_color(img, x1, y1, x2, y2) menerima parameter koordinat area sel pada image dan mengembalikan warna rata-rata, sedangkan color_similarity(c1, c2, threshold) membandingkan dua warna RGB untuk mendeteksi warna yang sama.

IMPLEMENTASI DALAM KODE

```
def load_image(self):
    filename = filedialog.askopenfilename(title="Pilih image",
                                          filetypes=[("Image files", "*.png *.jpg
*.jpeg *.bmp *.gif")])
    if filename:
        self.original_image = Image.open(filename).convert('RGB')
        self.grid_mode = True
        self.display_image_on_canvas()
        self.lbl_status.config(text="Image loaded. Set board size, click Apply,
then click image.", fg='#FFEAA7')

def display_image_on_canvas(self):
    if self.original_image is None:
        return
```

```
    img_ratio = self.original_image.width / self.original_image.height
    if img_ratio > 1:
        new_width = self.canvas_size
        new_height = int(self.canvas_size / img_ratio)
    else:
        new_height = self.canvas_size
        new_width = int(self.canvas_size * img_ratio)
    resized = self.original_image.resize((new_width, new_height),
Image.Resampling.LANCZOS)
    self.image_tk = ImageTk.PhotoImage(resized)
    self.canvas.delete("all")
    x_offset = (self.canvas_size - new_width) // 2
    y_offset = (self.canvas_size - new_height) // 2
    self.canvas.create_image(x_offset, y_offset, anchor=tk.NW,
image=self.image_tk, tags="image")
    self.image_bounds = (x_offset, y_offset, x_offset + new_width, y_offset +
new_height)

def extract_colors_from_image(self):
    if self.original_image is None or self.board_size == 0:
        return
    img = self.original_image
    img_cell_w = img.width / self.board_size
    img_cell_h = img.height / self.board_size
    board = []
    detected_colors = []
    char_counter = 0
    for r in range(self.board_size):
        row = []
        for c in range(self.board_size):
            x1, y1 = c * img_cell_w, r * img_cell_h
            x2, y2 = (c+1)*img_cell_w, (r+1)*img_cell_h
            avg_color = self.get_dominant_color(img, x1, y1, x2, y2)
            found_char = None
            for existing_color, existing_char in detected_colors:
                if self.color_similarity(avg_color, existing_color):
```

```
                    found_char = existing_char
                    break
            if found_char is None:
                found_char = chr(ord('A') + char_counter)
                detected_colors.append((avg_color, found_char))
                char_counter += 1
            row.append(found_char)
        board.append(row)
    self.extracted_colors = {char: color for color, char in detected_colors}
    self.solver.board = board
    self.solver.n = self.board_size
    self.assign_colors_from_extracted()
    self.draw_board()
```

3) Output sebagai Image

Program ini, menggunakan modul tkinter dan PIL untuk menyimpan hasil papan dengan peletakan queens sebagai file image, dengan fungsi save_solution_image() yang menerima input posisi queen dari solver.solution, membuat kanvas baru dalam memori menggunakan Image.new(), menggambar sel papan dengan warna sesuai color_map dan menambahkan lingkaran putih sebagai queen, kemudian menyimpan hasil sebagai file .png atau .jpg, fungsi ini menerima parameter file path dari filedialog.asksaveasfilename() dan tidak mengembalikan nilai, hanya menghasilkan file image yang menampilkan konfigurasi final.

IMPLEMENTASI DALAM KODE

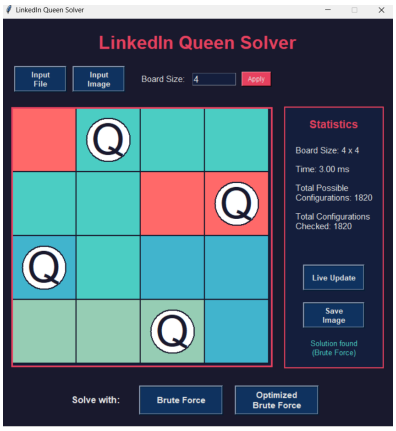
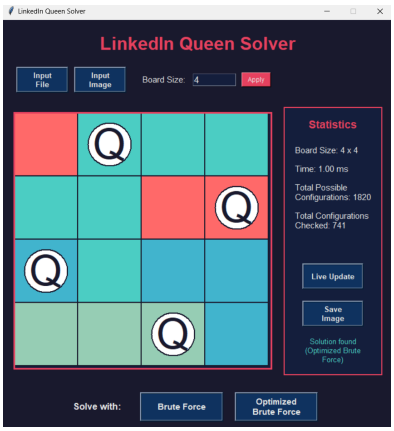
```
def save_solution_image(self):
    if self.solver.solution is None:
        messagebox.showwarning("Warning", "Tidak ada solusi untuk disimpan")
        return
    filename = filedialog.asksaveasfilename(title="Save Solution as Image",

initialfile="linkedin_queen_solution.png",
                                            defaultextension=".png",
                                            filetypes=[("PNG files", "*.png"),
("JPEG files", "*.jpg")])
    if filename:
        cell_size = 60
```

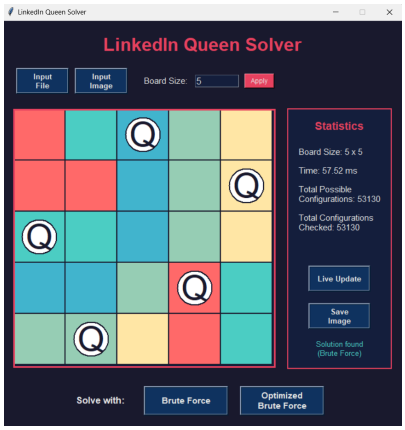
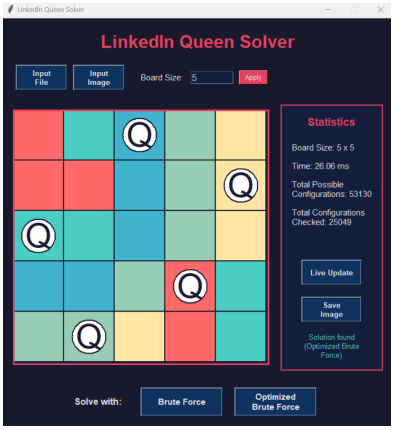
```
        padding = 20
        n = self.solver.n
        img_size = n*cell_size + 2*padding
        img = Image.new('RGB', (img_size, img_size), color='#1a1a2e')
        draw = ImageDraw.Draw(img)
        q_positions = set(self.solver.solution)
        for r in range(n):
            for c in range(n):
                x1 = padding + c*cell_size
                y1 = padding + r*cell_size
                x2 = x1 + cell_size
                y2 = y1 + cell_size
                char = self.solver.board[r][c]
                color = self.color_map.get(char, '#808080')
                draw.rectangle([x1, y1, x2, y2], fill=color, outline='#1a1a2e',
width=2)
                if (r,c) in q_positions:
                    circle_padding = cell_size//6
                    draw.ellipse([x1+circle_padding, y1+circle_padding,
                                  x2-circle_padding, y2-circle_padding],
                                 fill='white', outline='#1a1a2e', width=2)
                    try:
                        font = ImageFont.truetype("arial.ttf", cell_size//2)
                    except:
                        font = ImageFont.load_default()
                    text = "Q"
                    bbox = draw.textbbox((0,0), text, font=font)
                    text_w, text_h = bbox[2]-bbox[0], bbox[3]-bbox[1]
                    draw.text((x1+(cell_size-text_w)//2,
y1+(cell_size-text_h)//2-5),
                              text, fill='#1a1a2e', font=font)
        img.save(filename)
        messagebox.showinfo("Success", f"Solution saved to:\n{filename}")
```

BAB V
PENGUJIAN TEST CASE

- **Test Case 1 (File .txt)**

Input	Brute Force	Optimized Brute Force
	Output	Output
BAAA AABB CACC DDDC		
Statistik		
Waktu pencarian	3.0 ms	1.0 ms
Banyak kemungkinan	1820 kasus	1820 kasus
Banyak kasus yang ditinjau	1820 kasus	741 kasus

- **Test Case 2 (File .txt)**

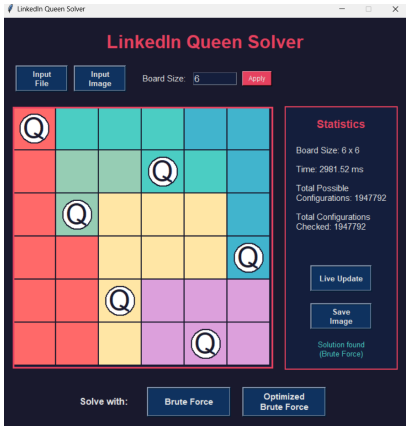
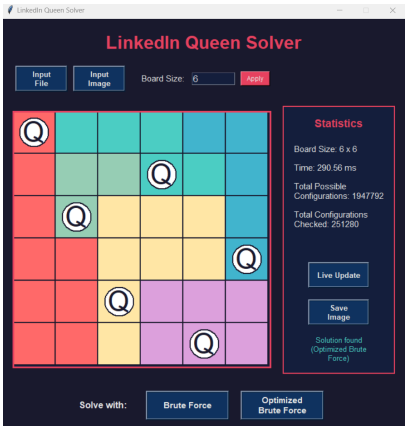
Input	Brute Force	Optimized Brute Force
	Output	Output
ABCDE AACDE BBCDE CCDAB DDEAB		
Statistik		
Waktu pencarian	57.52 ms	28.51 ms
Banyak kemungkinan	53130 kasus	53130 kasus
Banyak kasus yang ditinjau	53130 kasus	25049 kasus

- **Test Case 3 (File .txt)**

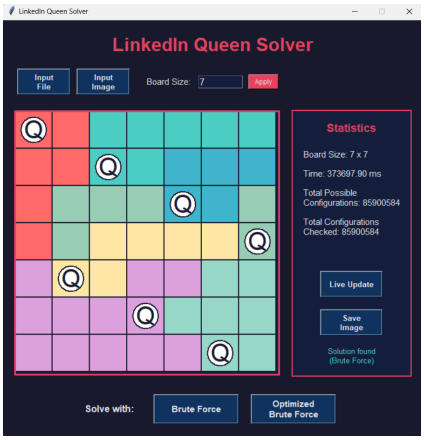
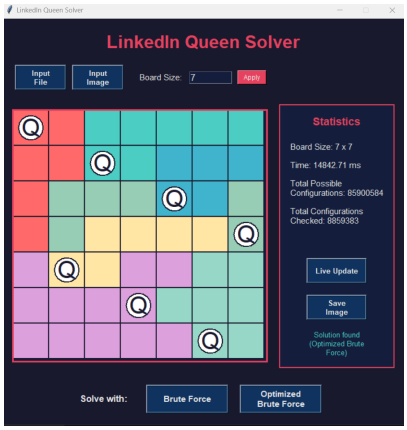
Input	Brute Force	Optimized Brute Force
	Output	Output
ABCDEF AABBCC DDEEFF ABCDEF CCDDAA EEFFBB		

Statistik		
Waktu pencarian	3074.91 ms	1159.23 ms
Banyak kemungkinan	1947792 kasus	1947792 kasus
Banyak kasus yang ditinjau	1947792 kasus	916594 kasus

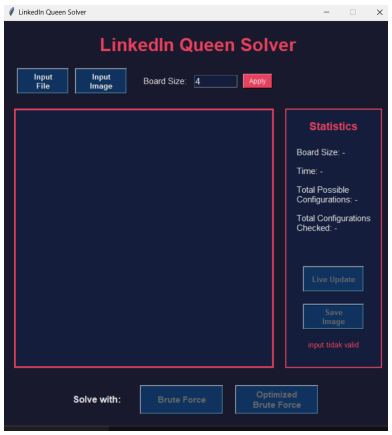
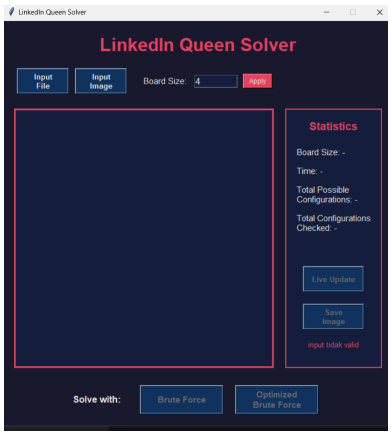
- **Test Case 4 (File .txt)**

Input	**Brute Force**	**Optimized Brute Force**
	Output	Output
CEEEBB CFFEEB CFAAAB CCAAAB CCADDD CCADDD		
Statistik		
Waktu pencarian	2981.52 ms	290.56 ms
Banyak kemungkinan	1947792 kasus	1947792 kasus
Banyak kasus yang ditinjau	1947792 kasus	251280 kasus

- **Test Case 5 (File .txt)**

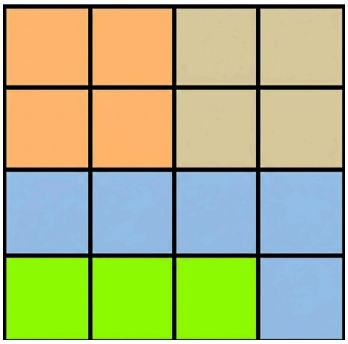
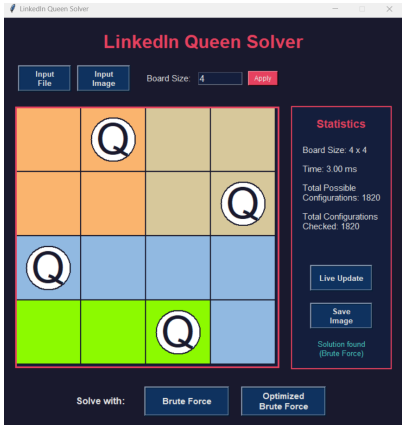
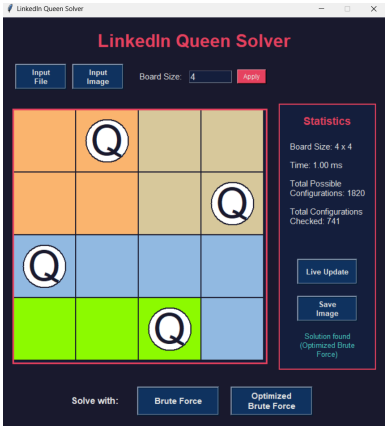
Input	Brute Force	Optimized Brute Force
	Output	Output
AABBBBB AABBCCC ADDDCCD ADEEEED FEEFFGG FFFFGGG FFFFFGG		
Statistik		
Waktu pencarian	373697.90 ms	14842.71 ms
Banyak kemungkinan	85900584 kasus	85900584 kasus
Banyak kasus yang ditinjau	85900584 kasus	8859383 kasus

- **Test Case 6 (File .txt) - Invalid Input Test**

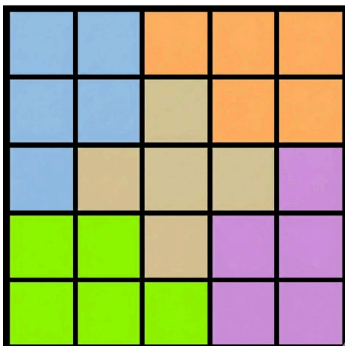
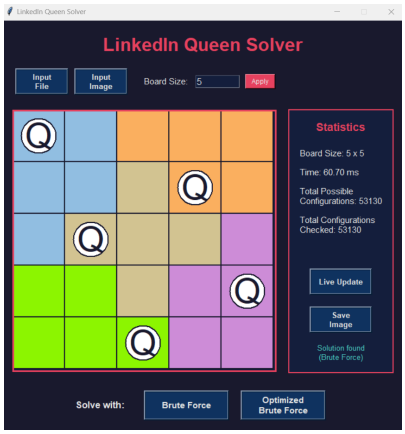
Input	Brute Force	Optimized Brute Force
	Output	Output
AAAA BBBB CCCC DDDD FFFF		

Statistik
Input invalid karena ukuran board bukan n x n dan jumlah warna melebihi jumlah n

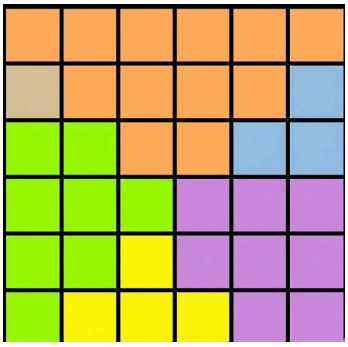
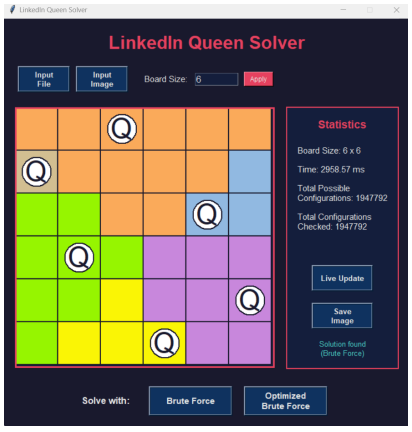
- **Test Case 7 (File .png)**

Input	Brute Force	Optimized Brute Force
	Output	Output
		
Statistik		
Waktu pencarian	3.0 ms	1.0 ms
Banyak kemungkinan	1820 kasus	1820 kasus
Banyak kasus yang ditinjau	1820 kasus	741 kasus

- **Test Case 8 (File .png)**

Input	Brute Force	Optimized Brute Force
	Output	Output
		
Statistik		
Waktu pencarian	60.70 ms	12.0 ms
Banyak kemungkinan	53130 kasus	53130 kasus
Banyak kasus yang ditinjau	53130 kasus	8508 kasus

- **Test Case 9 (File .png)**

Input	Brute Force	Optimized Brute Force
	Output	Output
		

Statistik		
Waktu pencarian	2958.57 ms	814.77 ms
Banyak kemungkinan	1947792 kasus	1947792 kasus
Banyak kasus yang ditinjau	1947792 kasus	717016 kasus

BAB VI
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	√	

Pernyataan tidak melakukan kecurangan:

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

Matthew Sebastian Kurniawan

Tautan Repositori: https://github.com/Matthew12-t/Tucil1_18223096.git