Laporan Tugas Kecil 1

IF2211 - Strategi Algoritma

Penyelesaian Permainan *Queens* Linkedin

Disusun oleh:

Nazwan Siddqi Muttaqin/18223066

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

A. Penjelasan Algoritma

Dalam menyelesaikan permainan *Queens* dari Linkedin ini, program menggunakan pendekatan algoritma *Brute Force* dengan teknik *Exhaustive Search* berbasis permutasi. Algoritma ini bekerja dengan cara men-*generate* seluruh kemungkinan konfigurasi penempatan queen pada board, kemudian melakukan pengujian dan validasi terhadap setiap konfigurasi tersebut untuk memverifikasi apakah seluruh aturan permainan terpenuhi.

Langkah - langkah algoritma:

1. Representasi Solusi

   Solusi direpresentasikan sebagai sebuah *array* satu dimensi $P = [c_0, c_1, \ldots, c_{n-1}]$, di mana indeks i menyatakan baris ke-i dan nilai c_i menyatakan posisi kolom dari *queen* pada baris tersebut. Representasi ini dipilih karena secara implisit telah memenuhi *constraint* bahwa setiap baris hanya boleh ditempati oleh satu *queen*, sehingga mengurangi ruang pencarian dibandingkan representasi matriks 2D murni.

2. *Generate* Permutasi

   Karena aturan permainan bahwa setiap baris dan kolom hanya boleh ditempati oleh satu *queen*, maka ruang solusi yang mungkin adalah himpunan permutasi dari angka kolom $[0, 1, \ldots, N-1]$. Algoritma men-*generate* permutasi kolom secara rekursif. Untuk papan berukuran $N \times N$, jumlah total permutasi yang akan diperiksa adalah sebanyak $N!$.

3. Validasi Konfigurasi

   Setiap permutasi yang di-*generate* akan divalidasi berdasarkan aturan khusus permainan *Queens*, yaitu:

   - *Color Constraint*: Memastikan tidak ada dua ratu atau lebih yang menempati wilayah warna yang sama. Program memetakan setiap sel (baris, kolom) ke warna tertentu sesuai input, dan memeriksa duplikasi warna pada konfigurasi saat ini.
   - *Neighbor/Touching Constraint*: Memastikan tidak ada dua ratu yang bersentuhan, baik secara horizontal, vertikal, maupun diagonal. Algoritma ini menerapkan optimasi dengan hanya membandingkan posisi *queen* pada baris ke-i dengan *queen* pada baris tepat di bawahnya $(i + 1)$, bukan dengan cara membandingkan seluruh pasangan *queen* dengan *nested loop*. Hal ini didasarkan pada prinsip bahwa dua *queen* dapat bersentuhan jika selisih barisnya adalah 1. Jika selisih posisi kolom antara baris i dan $i + 1$ kurang dari 2 ($|c_i - c_{i+1}| < 2$), maka kedua *queen* tersebut dinyatakan bersentuhan (baik secara vertikal maupun diagonal), sehingga konfigurasi dinyatakan tidak valid.

4. Mekanisme Pencarian

   Algoritma melakukan iterasi terhadap setiap permutasi yang dihasilkan. Jika ditemukan sebuah konfigurasi yang memenuhi seluruh kriteria validasi di atas, program akan langsung menghentikan pencarian dan mengembalikan konfigurasi, jumlah iterasi, dan waktu eksekusi tersebut sebagai solusi. Sebaliknya, jika seluruh kandidat solusi tidak ditemukan konfigurasi yang valid, program akan menyimpulkan bahwa tidak solusi tidak ada.

Kompleksitas waktu dari algoritma ini adalah $O(N.N!)$. Faktor $N!$ diperoleh dari banyaknya kemungkinan permutasi kolom yang di-*generate* sebagai ruang pencarian solusi. Sedangkan faktor N diperoleh dari fungsi cek_validasi, di mana algoritma melakukan iterasi tunggal sebanyak $N - 1$ kali untuk memeriksa constraint yang sudah didefinisikan.

B. *Source Code*

Berikut adalah implementasi kode program utama yang ditulis dalam bahasa Python. Kode dibagi menjadi kelas *QueensSolver* yang menangani logika algoritma dan kelas *QueensApp* yang menangani GUI.

class QueenSolver:

```
import customtkinter as ctk
from tkinter import filedialog, messagebox
from PIL import Image, ImageDraw, ImageFont
import time
import os
import re
import sys

ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")

class QueensSolver:
    def __init__(self, grid_warna):
        self.grid_warna = grid_warna
        self.n = len(grid_warna)
        self.solusi = None
```

```
        self.jumlah_percobaan = 0

    def generate_permutasi(self, elemen):
        if len(elemen) <= 1:
            yield elemen
        else:
            for permutasi in self.generate_permutasi(elemen[1:]):
                for i in range(len(elemen)):
                    yield permutasi[:i] + [elemen[0]] + permutasi[i:]

    def cek_validasi(self, posisi_queens):
        self.jumlah_percobaan += 1
        warna_terpakai = []

        # Constraint Warna
        for baris in range(self.n):
            kolom = posisi_queens[baris]
            warna = self.grid_warna[baris][kolom]

            if warna in warna_terpakai:
                return False
            warna_terpakai.append(warna)

        # Constraint Neighbor
        for i in range(self.n - 1):
            selisih_kolom = abs(posisi_queens[i] - posisi_queens[i+1])
            if selisih_kolom < 2:
                return False

        return True

    def cari_solusi(self):
        angka_kolom = list(range(self.n))
        generator = self.generate_permutasi(angka_kolom)
```

```
        for kemungkinan in generator:
            is_valid = self.cek_validasi(kemungkinan)
            if is_valid:
                yield kemungkinan, True, True
                return
            else:
                yield kemungkinan, False, False

        yield None, False, True
```

class QueensApp:

```
class QueensApp(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("Tucil 1 IF2211 - Queens Linkedin Solver")
        self.geometry("900x600")

        self.grid_data = []
        self.tombol_grid = {}
        self.solver = None
        self.generator_solusi = None
        self.start_time = 0
        self.is_running = False
        self.input_filename = None
        self.base_path = self.get_base_path()

        self.grid_columnconfigure(1, weight=1)
        self.grid_rowconfigure(0, weight=1)
        self.setup_sidebar()
        self.setup_board_area()
```

```
    def setup_sidebar(self):
        frame_kiri = ctk.CTkFrame(self, width=200, corner_radius=0)
        frame_kiri.grid(row=0, column=0, sticky="nsew")

        label_judul = ctk.CTkLabel(frame_kiri, text="Queens Linkedin
Solver\nBrute Force", font=("Roboto Medium", 20))
        label_judul.pack(pady=20, padx=5)

        self.btn_load = ctk.CTkButton(frame_kiri, text="Pilih File .txt",
command=self.load_file, font=("Roboto", 16))
        self.btn_load.pack(pady=10, padx=20)
        self.btn_start = ctk.CTkButton(frame_kiri, text="Mulai Cari Solusi",
command=self.start_solusi, state="disabled", font=("Roboto", 16))
        self.btn_start.pack(pady=10, padx=20)
        self.label_status = ctk.CTkLabel(frame_kiri, text="Menunggu file",
font=("Roboto", 14))
        self.label_status.pack(pady=(20, 5))
        self.label_iterasi = ctk.CTkLabel(frame_kiri, text="Iterasi: 0",
font=("Roboto", 14))
        self.label_iterasi.pack(pady=5)
        self.label_waktu = ctk.CTkLabel(frame_kiri, text="Waktu: 0 ms",
font=("Roboto", 14))
        self.label_waktu.pack(pady=5)

    def setup_board_area(self):
        self.frame_kanan = ctk.CTkFrame(self)
        self.frame_kanan.grid(row=0, column=1, sticky="nsew", padx=20, pady=20)

    def load_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Text files",
"*.txt")])
        if not file_path:
            return
```

```
        self.input_filename = os.path.basename(file_path)

        with open(file_path, 'r') as f:
            lines = f.readlines()

        self.grid_data = [list(line.strip()) for line in lines if line.strip()]
        self.n = len(self.grid_data)

        for line in self.grid_data:
            if len(line) != self.n:
                messagebox.showerror("Error", "File tidak valid: Grid harus
persegi (NxN).")
                return

        self.gambar_papan()
        self.label_status.configure(text=f"File berhasil dimuat
({self.n}x{self.n})")
        self.btn_start.configure(state="normal")


    def get_warna_region(self, huruf):
        warna_map = {
            'A': '#FFB5BA',
            'B': '#B5F0D5',
            'C': '#A8D8FF',
            'D': '#FFED99',
            'E': '#D9B3FF',
            'F': '#FFB380',
            'G': '#B8E994',
            'H': '#87E0D9',
            'I': '#FFA8CC',
            'J': '#C2A3FF',
            'K': '#FFDAA8',
            'L': '#FF9999',
```

```
            'M': '#99E6D9',
            'N': '#B3D9FF',
            'O': '#FFE5B4',
            'P': '#E6B3D6',
            'Q': '#D4FF99',
            'R': '#FFC2E0',
            'S': '#FFFAAA',
            'T': '#FFCC99',
            'U': '#C4E6E6',
            'V': '#D9D9D9',
            'W': '#A8E6CF',
            'X': '#B3B3FF',
            'Y': '#F5C2F5',
            'Z': '#99FFEB',
        }
        return warna_map.get(huruf.upper(), '#E8E8E8')

    def gambar_papan(self):
        for widget in self.frame_kanan.winfo_children():
            widget.destroy()

        self.tombol_grid = {}

        for i in range(self.n):
            self.frame_kanan.grid_rowconfigure(i, weight=1)
            self.frame_kanan.grid_columnconfigure(i, weight=1)

        for r in range(self.n):
            for c in range(self.n):
                huruf_region = self.grid_data[r][c]
                warna_bg = self.get_warna_region(huruf_region)

                kotak = ctk.CTkButton(self.frame_kanan, text=huruf_region,
fg_color=warna_bg, text_color=warna_bg, hover=False, corner_radius=0,
```

```
font=("Arial", 24, "bold"), border_width=0.5, border_color="black")
                kotak.grid(row=r, column=c, sticky="nsew")
                self.tombol_grid[(r, c)] = kotak

    def start_solusi(self):
        if not self.grid_data:
            return

        self.solver = QueensSolver(self.grid_data)
        self.generator_solusi = self.solver.cari_solusi()

        self.is_running = True
        self.start_time = time.time()
        self.btn_start.configure(state="disabled")
        self.btn_load.configure(state="disabled")
        self.label_status.configure(text="Mencari solusi..",
text_color="#75BAFE")

        self.update_logika()

    def update_logika(self):
        if not self.is_running:
            return

        try:
            steps_per_frame = 100
            data_terakhir = None
            found = False
            finished = False

            for _ in range(steps_per_frame):
                data_terakhir, found, finished = next(self.generator_solusi)
                if finished:
                    break
```

```
            waktu_berjalan = (time.time() - self.start_time) * 1000
            self.label_iterasi.configure(text=f"Iterasi:
{self.solver.jumlah_percobaan}")
            self.label_waktu.configure(text=f"Waktu: {waktu_berjalan:.2f} ms")

            if data_terakhir:
                self.visualisasi_queens(data_terakhir)

            if finished:
                self.is_running = False
                self.btn_load.configure(state="normal")
                self.btn_start.configure(state="normal")

                if found:
                    self.label_status.configure(text="Solusi ditemukan!",
text_color="green")
                    messagebox.showinfo("Sukses", f"Solusi ditemukan dalam
{self.solver.jumlah_percobaan} iterasi dan {waktu_berjalan:.2f} ms.")

                    simpan = messagebox.askyesno("Simpan Solusi", "Apakah Anda
ingin menyimpan solusi?")
                    if simpan:
                        self.simpan_solusi(data_terakhir, waktu_berjalan)
                else:
                    self.label_status.configure(text="Tidak ada solusi.",
text_color="red")
                    messagebox.showinfo("Info", "Algoritma selesai, tidak ada
solusi valid.")
            else:
                self.after(1, self.update_logika)

        except StopIteration:
            pass
```

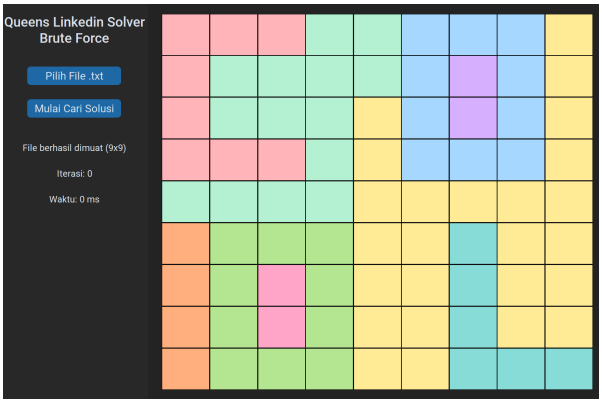
```
    def get_base_path(self):
        if getattr(sys, 'frozen', False):
            return os.path.dirname(os.path.dirname(sys.executable))
        else:
            return os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

    def simpan_solusi(self, posisi_queens, waktu_ms):
        test_folder = os.path.join(self.base_path, "test")
        os.makedirs(test_folder, exist_ok=True)

        nomor = "1"
        if self.input_filename:
            nums = re.findall(r'\d+', self.input_filename)
            if nums:
                nomor = nums[0]

        filepath_txt = os.path.join(test_folder, f"output{nomor}.txt")

        with open(filepath_txt, 'w') as f:
            for r in range(self.n):
                for c in range(self.n):
                    ada_queen = (c == posisi_queens[r])
                    if ada_queen:
                        f.write("#")
                    else:
                        huruf = self.grid_data[r][c]
                        f.write(huruf)
                f.write("\n")

        self.simpan_solusi_gambar(posisi_queens, nomor)

        messagebox.showinfo("Sukses", f"Solusi berhasil disimpan:\n-
{filepath_txt}\n- {os.path.join(test_folder, f'output{nomor}.png')}")
```

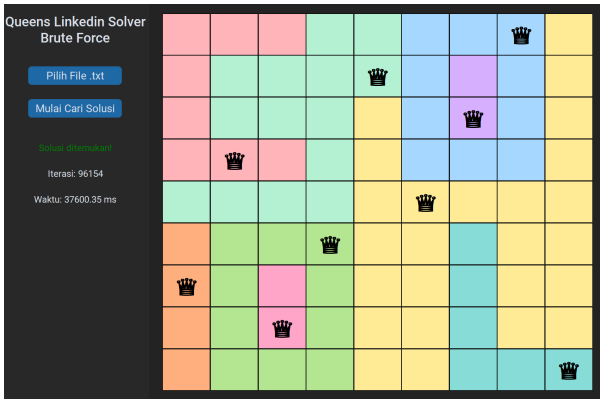
```
    def simpan_solusi_gambar(self, posisi_final, nomor):
        if not posisi_final:
            return

        test_folder = os.path.join(self.base_path, "test")
        os.makedirs(test_folder, exist_ok=True)

        cell_size = 100
        img_size = self.n * cell_size

        image = Image.new("RGB", (img_size, img_size), "white")
        draw = ImageDraw.Draw(image)
        font_queen = ImageFont.truetype("seguisym.ttf", size=int(cell_size *
0.6))

        for r in range(self.n):
            for c in range(self.n):
                x0 = c * cell_size
                y0 = r * cell_size
                x1 = x0 + cell_size
                y1 = y0 + cell_size

                huruf = self.grid_data[r][c]
                warna_hex = self.get_warna_region(huruf)

                draw.rectangle([x0, y0, x1, y1], fill=warna_hex,
outline="black", width=1)

                if posisi_final[r] == c:
                    center_x = x0 + (cell_size / 2)
                    center_y = y0 + (cell_size / 2)
                    draw.text((center_x, center_y), "♛", fill="black",
font=font_queen, anchor="mm")
```

```
        filepath = os.path.join(test_folder, f"output{nomor}.png")
        image.save(filepath)

    def visualisasi_queens(self, posisi_queens):
        for (r, c), tombol in self.tombol_grid.items():
            tombol.configure(text="")

        for r, c in enumerate(posisi_queens):
            tombol = self.tombol_grid[(r, c)]
            tombol.configure(text="♛", text_color="black", font=("Segoe UI
Symbol", 36, "bold"))

if __name__ == "__main__":
    app = QueensApp()
    app.mainloop()
```

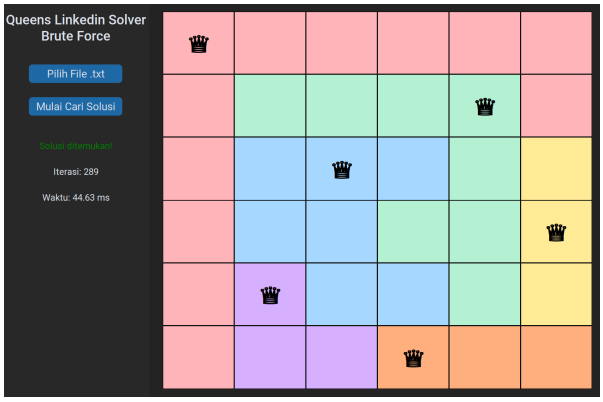
C. Hasil Program

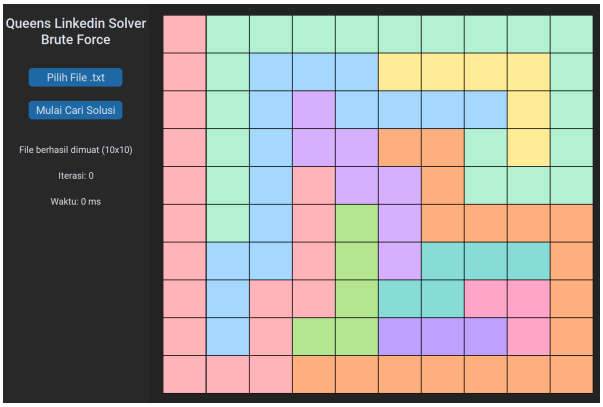
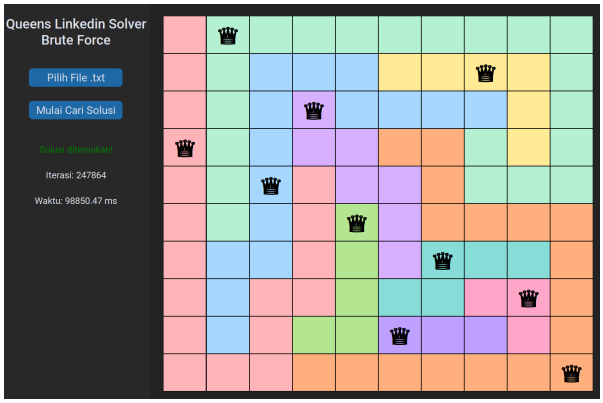
Bagian ini memaparkan hasil eksekusi program terhadap beberapa *test cases* yang berbeda ukurannya. Setiap pengujian menampilkan visualisasi papan awal, visualisasi solusi akhir, serta data kinerja algoritma berupa waktu eksekusi dan jumlah iterasi.

<table>
<tr><td colspan="2">Percobaan 1</td></tr>
<tr><td>Input</td><td>Output</td></tr>
<tr><td> 

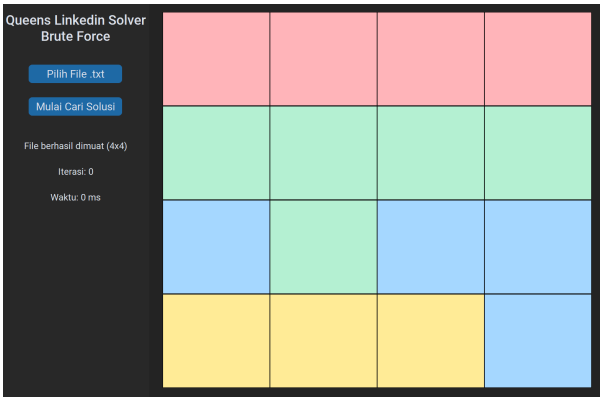
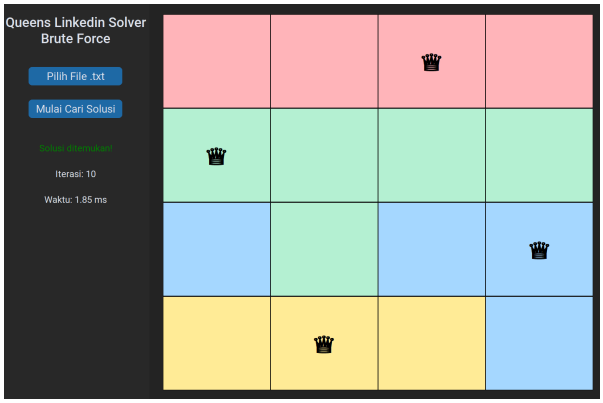
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH</td><td> 

AAABBCC#D
ABBB#CECD
ABBBDC#CD
A#ABDCCCD
BBBBD#DDD
FGG#DDHDD
#GIGDDHDD
FG#GDDHDD
FGGGDDHH#</td></tr>
<tr><td colspan="2">Iterasi : 96154</td></tr>
<tr><td colspan="2">Waktu : 37600.35 ms</td></tr>
</table>

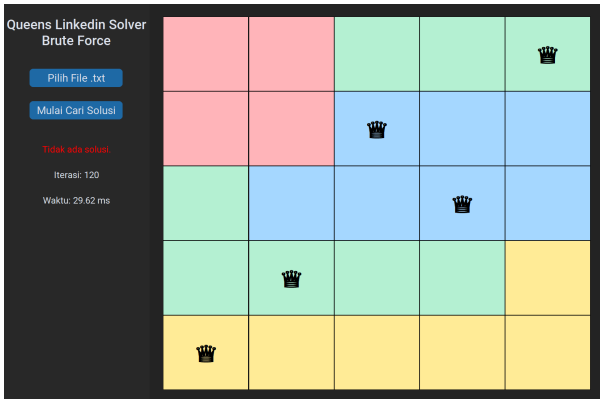
Percobaan 2	
Input	Output
 AAAAAA ABBBBA ACCCBD ACCBBD AECCBD AEEFFF	 #AAAAA ABBB#A AC#CBD ACCBB# A#CCBD AEE#FF
Iterasi : 289	
Waktu : 344.63 ms	

Percobaan 3	
Input	Output
 ABBBBBBBBB ABCCCDDDDB ABCECCCCDB	 A#BBBBBBBB ABCCCDD#DB ABC#CCCCDB

ABCEEFFBDB ABCAEEFBBB ABCAGEFFFF ACCAGEHHHF ACAAGHHIIF ACAGGJJJIF AAAFFFFFFF	#BCEEFFBDB AB#AEEFBBB ABCA#EFFFF ACCAGE#HHF ACAAGHHI#F ACAGG#JJIF AAAFFFFFF#
Iterasi : 247864	
Waktu : 98850.47 ms	

Percobaan 4	
Input	Output
 AAAA BBBB CBCC DDDC	 AA#A #BBB CBC# D#DC
Iterasi : 10	
Waktu : 1.85 ms	

<table>
<tr><td colspan="2">Percobaan 5</td></tr>
<tr><td>Input</td><td>Output</td></tr>
<tr><td> 

AABBB
AACCC
BCCCC
BBBBD
DDDDD</td><td> 

Tidak ada solusi</td></tr>
<tr><td colspan="2">Iterasi : 120 (Tidak ada solusi)</td></tr>
<tr><td colspan="2">Waktu : 29.62 ms (Tidak ada solusi)</td></tr>
</table>

D. *Link Repository*

Kode program lengkap beserta instruksi penggunaan dan *file executable* dapat diakses melalui *link repository* berikut:

NazwanSM/Tucil1_18223066

E. Surat Pernyataan

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

Nazwan Siddqi Muttaqin

F. 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	✓	