LAPORAN TUGAS KECIL 1

IF2211 STRATEGI ALGORITMA

Disusun oleh:

Helena Kristela Sarhawa

13524109

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2026

PENDAHULUAN

Queens adalah gim logika yang tersedia pada situs jejaring profesional LinkedIn. Tujuan dari gim ini adalah menempatkan queen pada sebuah papan persegi berwarna sehingga terdapat hanya satu queen pada tiap baris, kolom, dan daerah warna. Selain itu, satu queen tidak dapat ditempatkan bersebelahan dengan queen lainnya, termasuk secara diagonal. (Sumber: Deskripsi Tugas Kecil 1 IF2211 Strategi Algoritma Semester II tahun 2025/2026)

PENJELASAN ALGORITMA *BRUTE FORCE*

Mengacu pada bahan kuliah Algoritma *Brute Force* (Bagian 1) (Versi update 2026), algoritma *brute force* merupakan algoritma yang menyelesaikan persoalan secara lempang (*straightforward*). Pada persoalan Queens ini, algoritma *brute force* digunakan untuk mencari solusi yang pertama kali ditemukan dari semua kemungkinan jawaban yang ada. Program yang dirancang dapat menghasilkan dua jenis jawaban, yakni solusi ditemukan atau solusi tidak ada. Program juga didesain agar *input* persoalan dinyatakan valid terlebih dahulu sebelum bisa dicari solusinya. Definisi valid yang dimaksud yakni ukuran kolom sama dengan ukuran baris, karakter yang digunakan hanya "A-Z", dan jumlah daerah warna pada *input* sama dengan ukuran kolom/baris. *Input* yang diterima program berupa *file* .txt dengan *output* yang dihasilkan dapat disimpan dalam *file* .txt atau .png untuk memperoleh gambaran persoalan secara visual. *Output* .png hanya dapat diperoleh jika persoalan memiliki solusi dan gambar hanya bisa disimpan dari tampilan GUI.

Dalam program ini, posisi queens disimpan dalam bentuk list yang berisikan pasangan nilai (baris, kolom) dengan contoh: queen = [(0, c0), (1, c1), (2, c2), ..., (N-1, cN-1)] yang berarti queen berada pada baris ke-0 kolom c0, baris ke-1 kolom c1, dan seterusnya. Dengan begitu, total kemungkinan terburuk yang diperiksa adalah N! dengan N adalah ukuran baris/kolom *board* persoalan.

Algoritma *brute force* diterapkan ke dalam beberapa langkah. Pertama, *input file* .txt akan divalidasi sesuai definisi valid yang tertera di atas. Jika *input* tidak valid, program akan berhenti dan pencarian solusi tidak dapat dilakukan. Selanjutnya, semua kemungkinan posisi queen akan dibangun menggunakan permutasi kolom, dengan mula-mula dibuatnya permutasi awal [0, 1, 2, …, N-1]. Setiap permutasi kolom akan

dihasilkan satu per satu dari permutasi awal menggunakan fungsi bangun_permutasi(). Setelah itu, setiap permutasi digunakan untuk membentuk posisi queen = [(0, perm[0]), (1, perm[1]), ..., (N-1, perm[N-1])]. Setiap kemungkinan posisi queen akan dicek menggunakan fungsi cek_daerah() untuk memastikan hanya ada satu queen di daerah warna yang sama, cek_tetangga() untuk memastikan tidak ada queen yang berdekatan, dan cek_kolom_baris() untuk memastikan tidak ada queen yang berada di kolom dan baris yang sama. Jika ditemukan satu kemungkinan yang memenuhi ketiga fungsi pengecekan, pencarian solusi akan dihentikan dan kemungkinan tersebut menjadi solusi akhir persoalan. Jika setelah semua kemungkinan dicek dan tidak ada yang memenuhi ketiga fungsi, maka persoalan dikatakan tidak memiliki solusi. Setelah pencarian solusi berhenti, waktu pencarian dan jumlah kemungkinan yang dicek akan ditampilkan. Waktu pencarian hanya meliputi waktu pencarian solusi menggunakan algoritma *brute force* tanpa memasukkan waktu pembacaan dan pengecekan validasi *input* dan penyimpanan *output*. Algoritma *brute force* ini memiliki kompleksitas O(N!) karena pengecekan kemungkinan dilakukan pada semua permutasi kolom.

SOURCE PROGRAM

Program dibuat menggunakan bahasa pemrograman Python dengan antarmuka CLI dan GUI menggunakan PyQt6. Berikut merupakan kode program yang menjadi inti logika pemecahan persoalan menggunakan algoritma *brute force*.

```
def parsing(lines):
    board = []
    for baris in lines:
        line = baris.strip()
        if line == "" : continue
        if " " in line:
            hasil = line.split()
        else:
            hasil = list(line)
        board.append(hasil)
    return board
```

```
def validasi_input(board):
    n = len(board)
    if n == 0:
        return False
    for row in board:
        if len(row) != n:
            return False
    for b in range(n):
        for k in range(n):
            now = board[b][k]
            if len(now) != 1 or not('A' <= now <= 'Z'):
                return False
    if n > 26:
        return False
    daerah = set_daerah(board)
    if len(daerah) != n:
        return False
    return True

def set_daerah(board):
    n = len(board)
    s = set()
    for b in range(n):
        for k in range(n):
            s.add(board[b][k])
    return s

def cek_daerah(board, queen):
    daerah = set()
    for (b,k) in queen:
        now = board[b][k]
        if now in daerah: return False
        daerah.add(now)
    return True
```

```
def cek_tetangga(queen):
    posisi_queen = set(queen)
    for b, k in queen:
         if (b-1, k-1) in posisi_queen or (b-1, k) in posisi_queen or (b-1, k+1) in posisi_queen or (b, k-1) in posisi_queen or (b, k+1) in posisi_queen or (b+1, k-1) in posisi_queen or (b+1, k) in posisi_queen or (b+1, k+1) in posisi_queen: return False
    return True

def cek_kolom_baris(queen, n):
    if len(queen) != n : return False
    baris = set()
    kolom = set()
    for (b,k) in queen:
        if b in baris or k in kolom: return False
        baris.add(b)
        kolom.add(k)
    return True

def bangun_permutasi(array):
    i = len(array) - 2
    while i>=0 and array[i]>=array[i+1]: i -= 1
    if i < 0 : return False
    j = len(array) - 1
    while array[j]<=array[i]: j -= 1
    temp = array[i]
    array[i] = array[j]
    array[j] = temp
    kiri = i + 1
    kanan = len(array) - 1
    while kiri < kanan:
        temp = array[kiri]
        array[kiri] = array[kanan]
        array[kanan] = temp
        kiri += 1
        kanan -= 1
    return True
```

```
def semua_kemungkinan(n):
    permutasi = list(range(n))
    yield permutasi[:]
    while bangun_permutasi(permutasi): yield permutasi[:]

def langkah_bruteforce(board, skip_time=False):
    if not skip_time and not validasi_input(board):
        yield 0, None, False
        return
    n = len(board)
    kasus = 0
    for permutasi in semua_kemungkinan(n):
        kasus += 1
        queen = [(r,permutasi[r]) for r in range(n)]
              valid = cek_daerah(board,queen) and cek_tetangga(queen) and
cek_kolom_baris(queen,n)
        yield kasus, queen, valid
        if valid: return

def hasil_bruteforce(board, skip_time=False):
    if not skip_time and not validasi_input(board): return None, 0
    n = len(board)
    kasus = 0
    for permutasi in semua_kemungkinan(n):
        kasus += 1
        queen = [(r,permutasi[r]) for r in range(n)]
                  if cek_daerah(board,queen) and cek_tetangga(queen) and
cek_kolom_baris(queen,n): return queen, kasus
    return None, kasus

def print_board(board,queen):
    copy_board = [baris[:] for baris in board]
    if queen is not None:
        for (b,k) in queen: copy_board[b][k] = '#'
    hasil = []
    for baris in copy_board: hasil.append(''.join(baris))
```

```
    return hasil
```

Berikut juga terlampir *source code* program untuk menampilkan *output* dalam format CLI.

```
import time, os
from bruteforce import parsing, langkah_bruteforce, print_board, validasi_input

print("Masukkan path file input (.txt)!")
print("Contoh: ../test/tc1.txt")
input_path = input("Path file: ").strip()

if not os.path.exists(input_path):
    print("File tidak ditemukan")
    exit()

with open(input_path, "r", encoding="utf-8") as f: lines = f.readlines()

board = parsing(lines)
if not validasi_input(board):
    print("Board tidak valid")
    exit()
UPDATE = 100
solusi = None
last_kasus = 0
log_lines = []
start = time.perf_counter()
steps = langkah_bruteforce(board, skip_time=True)

for kasus, queen, valid in steps:
    last_kasus = kasus
    if valid:
        solusi = queen
        print("Solusi ditemukan di kasus:", kasus)
        break
    if kasus % UPDATE == 0:
```

```
        print("Progress (kasus =", kasus, "):")
        for baris in print_board(board,queen):
            print(baris)
        print()
end = time.perf_counter()
total_waktu = (end - start) * 1000
if solusi is None:
    print("\nTidak ada solusi")
else:
    for baris in print_board(board,solusi):
        print(baris)
        log_lines.append(baris)
print()
print(f"Waktu eksekusi: {total_waktu:.2f} ms")
print(f"Banyak kasus yang ditinjau: {last_kasus} kasus")
jawab = input("\nSimpan hasil ke file .txt? (y/n): ").strip().lower()
if jawab == "y":
    output_path = input("Nama file output (contoh: solusi.txt): ").strip()
            with open(output_path, "w", encoding="utf-8") as f:
f.write("\n".join(log_lines))
    print(f"Hasil disimpan ke file: {output_path}")
else: print("Hasil tidak disimpan")
```

Sebagai soal bonus, terlampir juga *source code* program untuk menampilkan *output* dalam format GUI menggunakan PyQt6.

```
import sys, time
from PyQt6.QtCore import QThread, pyqtSignal, Qt
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QTextEdit, QLabel, QFileDialog, QMessageBox, QSpinBox
from PyQt6.QtGui import QPixmap, QPainter, QColor, QFont
from bruteforce import parsing, langkah_bruteforce, print_board, validasi_input

def build_color(board_lines):
    symbols = sorted({ch for row in board_lines for ch in row if ch != '#'})
    m = len(symbols)
```

```
    palette = {}
    if m == 0: return palette
    for i, sym in enumerate(symbols):
        hue = int(360 * i / m)
        sat = 150
        val = 235
        palette[sym] = QColor.fromHsv(hue, sat, val)
    return palette

def cell_size(n, max_canvas=1000, margin=20, min_cell=6, max_cell=60):
    usable = max_canvas - 2 * margin
    cell = usable // max(1, n)
    return max(min_cell, min(max_cell, cell))

def board_image(base_lines, queen_lines, max_canvas=1000, margin=20):
    n = len(base_lines)
    cell = cell_size(n, max_canvas=max_canvas, margin=margin)
    w = margin * 2 + n * cell
    h = margin * 2 + n * cell
    pix = QPixmap(w, h)
    pix.fill(QColor("#FFF7ED"))
    palette = build_color(base_lines)
    p = QPainter(pix)
    font = QFont("Segoe UI Symbol", max(8, int(cell * 0.65)))
    font.setBold(True)
    p.setFont(font)
    for r in range(n):
        for c in range(n):
            x = margin + c * cell
            y = margin + r * cell
            ch_base = base_lines[r][c]
            fill = palette.get(ch_base, QColor("#FFFFFF"))
            p.fillRect(x, y, cell, cell, fill)
            p.setPen(QColor("#3B1D1A"))
            p.drawRect(x, y, cell, cell)
            if queen_lines[r][c] == '#':
```

```
                p.setPen(QColor("#111111"))
                   p.drawText(x, y, cell, cell, Qt.AlignmentFlag.AlignCenter,
"#")
    p.end()
    return pix

class SolverThread(QThread):
    progress = pyqtSignal(int,list)
    finished_ok = pyqtSignal(int,list)
    finished_fail = pyqtSignal(int,list)
    error = pyqtSignal(str)

    def __init__(self,board,update_every=10,parent=None):
        super().__init__(parent)
        self.board = board
        self.update_every = update_every
    def run(self):
        try:
            start = time.perf_counter()
            steps = langkah_bruteforce(self.board, skip_time=True)
            solusi = None
            last_kasus = 0
            last_lines = print_board(self.board, None)
            for kasus, queen, valid in steps:
                last_kasus = kasus
                last_lines = print_board(self.board,queen)
                if valid:
                    end = time.perf_counter()
                    total_waktu = (end - start) * 1000
                    solusi = queen
                    solusi_lines = print_board(self.board, solusi)
                        self.finished_ok.emit(kasus, solusi_lines + [f"Waktu
eksekusi: {total_waktu:.2f} ms"])
                    return
                                    if kasus % self.update_every == 0:
self.progress.emit(kasus,last_lines)
            end = time.perf_counter()
```

```
            total_waktu = (end - start) * 1000
                  self.finished_fail.emit(last_kasus, last_lines + [f"Waktu
eksekusi: {total_waktu:.2f} ms"])
        except Exception as e: self.error.emit(str(e))


class Main(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Queens Solver (Tucil1_13524109)")
        self.resize(900,650)
        self.input_path = None
        self.board = None
        self.worker = None
        self.last_result_text = ""
        self.last_result_board_lines = None

        central = QWidget()
        self.setCentralWidget(central)
        root = QVBoxLayout(central)

        top = QHBoxLayout()
        self.btn_open = QPushButton("Open File .txt")
        self.btn_run = QPushButton("Run")
        self.btn_save = QPushButton("Save Result .txt")
        self.btn_save_img = QPushButton("Save Image")
        self.btn_run.setEnabled(False)
        self.btn_save.setEnabled(False)
        self.btn_save_img.setEnabled(False)

        top.addWidget(self.btn_open)
        top.addWidget(self.btn_run)
        top.addWidget(self.btn_save)
        top.addWidget(self.btn_save_img)
        top.addStretch(1)

        top.addWidget(QLabel("Update setiap"))
```

```
        self.spin_update = QSpinBox()
        self.spin_update.setRange(1, 10_000_000)
        self.spin_update.setValue(10)
        top.addWidget(self.spin_update)
        top.addWidget(QLabel("kasus"))
        root.addLayout(top)

        self.lbl_status = QLabel("Belum ada file yang dibuka")
        root.addWidget(self.lbl_status)

        self.text = QTextEdit()
        self.text.setReadOnly(True)
        self.text.setPlaceholderText("Live Update")
        root.addWidget(self.text, 1)

        self.btn_open.clicked.connect(self.open_file)
        self.btn_run.clicked.connect(self.run_solver)
        self.btn_save.clicked.connect(self.save_output)
        self.btn_save_img.clicked.connect(self.save_image)

    def open_file(self):
          path, _= QFileDialog.getOpenFileName(self, "Pilih File Input", "",
"Text Files (*.txt);;All Files (*)")
        if not path: return
        try:
                    with open(path, "r", encoding="utf-8") as f: lines =
f.readlines()
            board = parsing(lines)
            if not validasi_input(board):
                self.input_path = path
                self.board = None
                preview = "\n".join(print_board(board, None))
                self.text.setPlainText(preview)
                self.lbl_status.setText("Board tidak valid")
                self.btn_run.setEnabled(False)
                self.btn_save.setEnabled(False)
                self.last_result_text = ""
```

```
                self.last_result_board_lines = None
                self.btn_save_img.setEnabled(False)
                  QMessageBox.warning(self, "Input tidak valid", "Board tidak valid!\n\n" "Syarat:\n" "- Ukuran NxN\n" "- Hanya mengandung A-Z\n" "- Jumlah daerah (region warna) = N\n" "- N <= 26\n")
                return
            self.input_path = path
            self.board = board
            preview = "\n".join(print_board(board,None))
            self.text.setPlainText(preview)
            self.lbl_status.setText(f"File dibuka: {path}")
            self.btn_run.setEnabled(True)
            self.btn_save.setEnabled(False)
            self.last_result_text = ""
            self.btn_save_img.setEnabled(False)
            self.last_result_board_lines = None
          except Exception as e: QMessageBox.critical(self, "Error", f"Gagal Membuka File:\n{e}")


    def run_solver(self):
        if self.board is None:
              QMessageBox.warning(self, "Warning", "Buka file input terlebih dahulu")
            return
        self.btn_run.setEnabled(False)
        self.btn_open.setEnabled(False)
        self.btn_save.setEnabled(False)
        self.btn_save_img.setEnabled(False)
        update_every = self.spin_update.value()
        self.lbl_status.setText("Running...")
        self.worker = SolverThread(self.board, update_every=update_every)
        self.worker.progress.connect(self.on_progress)
        self.worker.finished_ok.connect(self.on_finished_ok)
        self.worker.finished_fail.connect(self.on_finished_fail)
        self.worker.error.connect(self.on_error)
        self.worker.start()
```

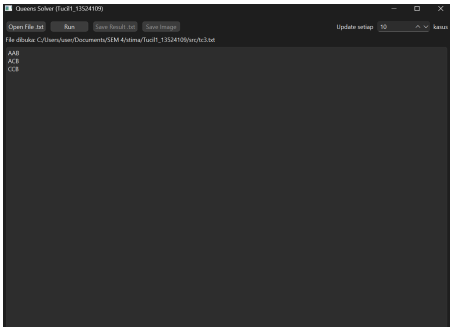
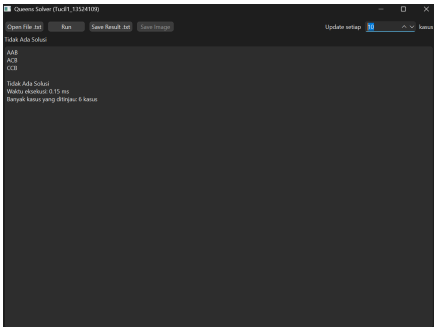
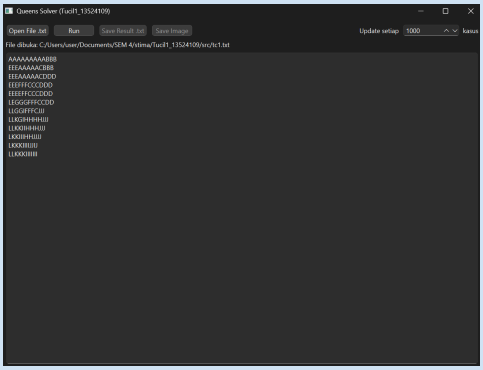
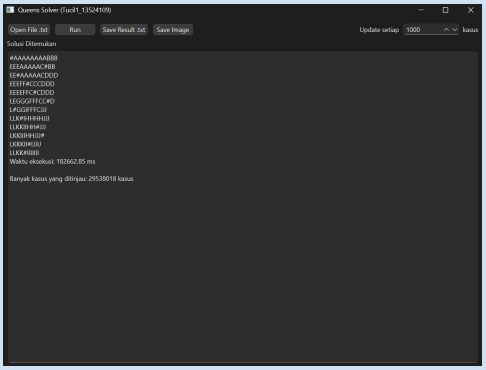
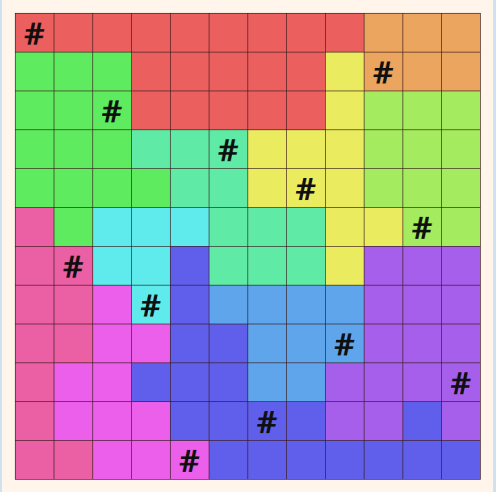
```
    def on_progress(self, kasus, board_lines):
        text = []
        text.append(f"Progress ({kasus} kasus)")
        text.extend(board_lines)
        text.append("")
        self.text.setPlainText("\n".join(text))

    def on_finished_ok(self, kasus, solusi_lines):
        self.btn_open.setEnabled(True)
        self.btn_run.setEnabled(True)
        self.btn_save.setEnabled(True)
        message = []
        message.extend(solusi_lines)
        message.append("")
        message.append(f"Banyak kasus yang ditinjau: {kasus} kasus")
        out = "\n".join(message)
        self.text.setPlainText(out)
        self.lbl_status.setText("Solusi Ditemukan")
        self.last_result_text = out
        self.last_result_board_lines = solusi_lines
        self.btn_save_img.setEnabled(True)
        self.ask_save()
        self.worker = None

    def on_finished_fail(self, kasus, last_lines):
        self.btn_open.setEnabled(True)
        self.btn_run.setEnabled(True)
        self.btn_save.setEnabled(True)
        awal_lines = print_board(self.board, None)
        message = []
        message.extend(awal_lines)
        message.append("")
        message.append(f"Tidak Ada Solusi")
            if last_lines and last_lines[-1].startswith("Waktu eksekusi:"):
message.append(last_lines[-1])
        message.append(f"Banyak kasus yang ditinjau: {kasus} kasus")
```

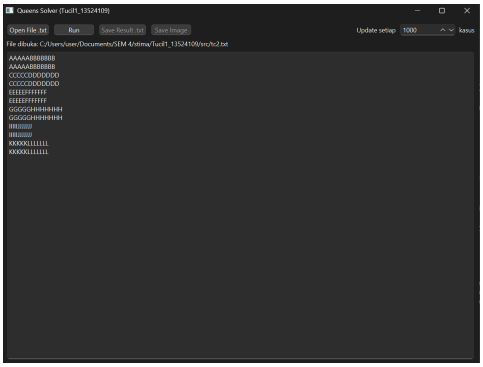
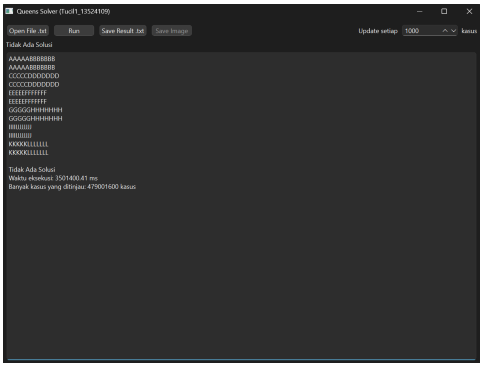
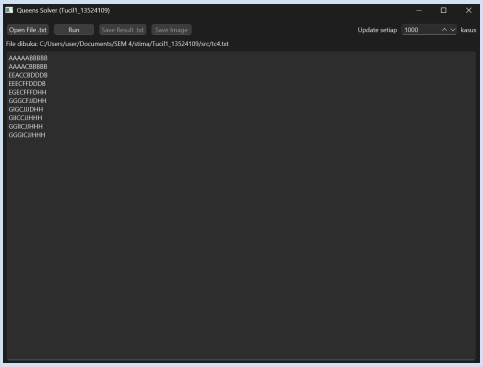
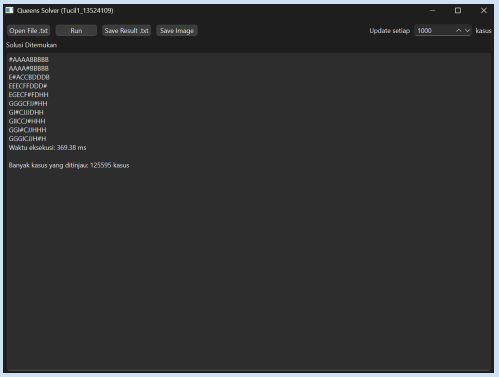
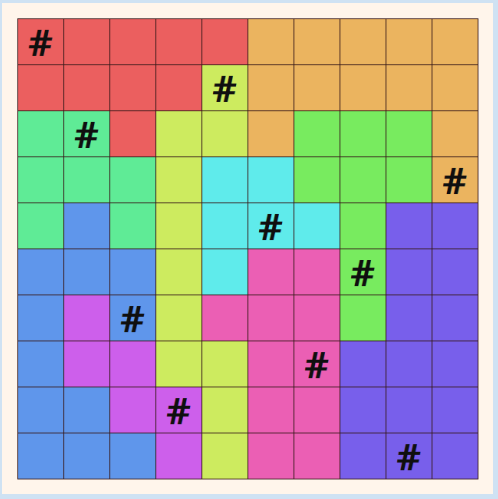
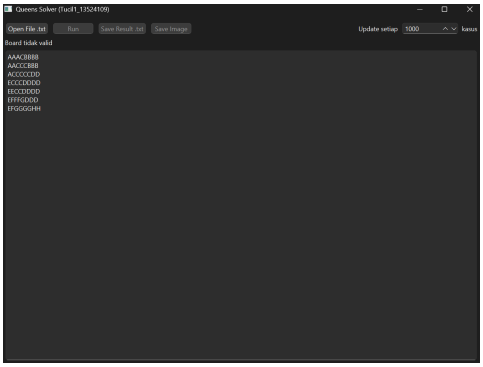
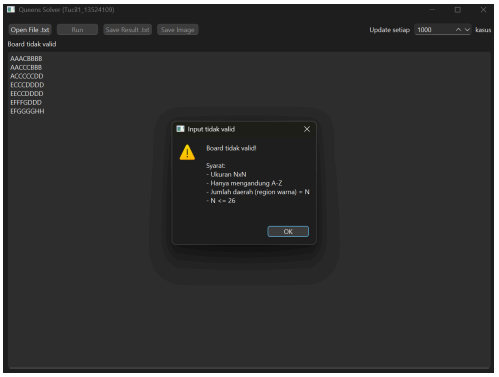
```
        out = "\n".join(message)
        self.text.setPlainText(out)
        self.lbl_status.setText("Tidak Ada Solusi")
        self.last_result_text = out
        self.last_result_board_lines = None
        self.btn_save_img.setEnabled(False)
        self.ask_save()
        self.worker = None

    def on_error(self, message):
        self.btn_open.setEnabled(True)
        self.btn_run.setEnabled(True)
        self.btn_save.setEnabled(False)
        self.last_result_board_lines = None
        self.btn_save_img.setEnabled(False)
        QMessageBox.critical(self, "Error", f"Terjadi error:\n{message}")
        self.worker = None

    def ask_save(self):
        if not self.last_result_text: return
          answer = QMessageBox.question(self, "Simpan hasil?", "Ingin simpan
hasil    ke    file    .txt    baru?",    QMessageBox.StandardButton.Yes    |
QMessageBox.StandardButton.No)
        if answer == QMessageBox.StandardButton.Yes: self.save_output()

    def save_output(self):
        if not self.last_result_text:
                QMessageBox.information(self, "Info", "Belum ada hasil untuk
disimpan")
            return
         path, _ = QFileDialog.getSaveFileName(self, "Save as", "solusi.txt",
"Text Files (*.txt);;All Files (*)")
        if not path: return
        try:
                            with  open(path,  "w",  encoding="utf-8")  as  f:
f.write(self.last_result_text)
```

```
                QMessageBox.information(self, "Berhasil", f"Hasil disimpan
ke:\n{path}")
          except Exception as e: QMessageBox.critical(self, "Error", f"Gagal
menyimpan:\n{e}")


    def save_image(self):
        if not self.last_result_board_lines:
               QMessageBox.information(self, "Info", "Belum ada solusi untuk
disimpan sebagai gambar.")
            return
              path, _ = QFileDialog.getSaveFileName(self, "Save image as",
"solusi.png", "PNG Image (*.png)")
        if not path: return
        try:
                        pix = board_image(print_board(self.board, None),
self.last_result_board_lines, max_canvas=1200, margin=20)
            ok = pix.save(path, "PNG")
            if not ok: raise RuntimeError("Gagal menyimpan PNG.")
                 QMessageBox.information(self, "Berhasil", f"Gambar disimpan
ke:\n{path}")
        except Exception as e:
                    QMessageBox.critical(self, "Error", f"Gagal menyimpan
gambar:\n{e}")


def main():
    app = QApplication(sys.argv)
    w = Main()
    w.show()
    sys.exit(app.exec())


if __name__ == "__main__":
    main()
```

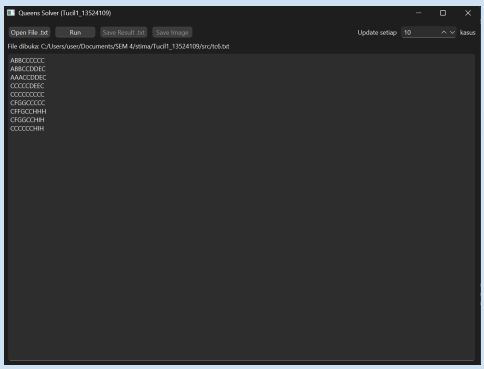
Untuk menjalankan program, ketik cd src lalu python main.py jika ingin *output* CLI. Jika ingin *output* GUI, ketik python gui.py.

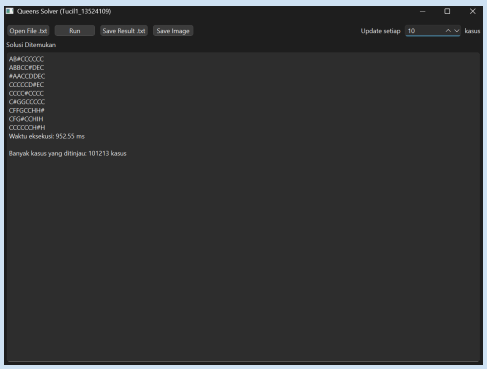
TEST CASE

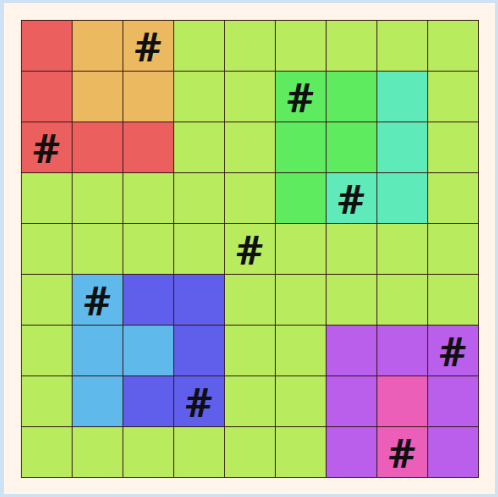
No.	*Input*	*Output*
1.		
2.		 

3.		
4.		 
5.		 keterangan: tidak valid karena ukuran *input* tidak N x N

6.







Berikut adalah tautan ke *repository* Tucil1_13524109: https://github.com/helenakristela/Tucil1_13524109

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

Helena Kristela Sarhawa

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	✓	