Laporan Tugas Kecil 1 IF2211 Strategi Algoritma
Penyelesaian Permainan *Queens* Linkedin

Surya Suharna
18223075
18223075@std.stei.itb.ac.id

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2026

DAFTAR ISI

DEFINISI MASALAH

Permainan ***Queens*** Linkedin adalah sebuah permainan yang bertujuan untuk menempatkan sebanyak n buah *queen* pada sebuah papan dengan n x n berwarna. Terdapat beberapa hal yang perlu diperhatikan pada persoalan ini yaitu.

1. Setiap queen hanya boleh menempati satu baris (*row).*
2. Setiap queen hanya boleh menempati satu kolom (c*olumn*).
3. Setiap warna hanya boleh ditempati tepat satu queen.
4. Setiap queen tidak boleh ditempatkan bersebelahan dengan queen lainnya, termasuk secara diagonal.

Pemecahan masalah yang perlu dilakukan pada konteks ini adalah menemukan satu solusi penempatan queen pada papan berwarna yang diberikan atau menampilkan sebuah keluaran bahwa permasalahan tersebut tidak memiliki solusi. Dalam pengerjaannya pencarian solusi dilakukan menggunakan algoritma ***brute force.***

DESAIN SOLUSI

Berdasarkan definisi masalah di atas berikut bagaimana saya melakukan desain solusi dari aturan yang berlaku pada gim ini.

1. **Definisi *block***

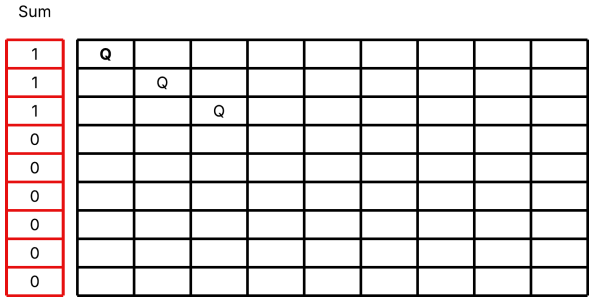
   Satu kotak akan disebut satu *block* yang mana *block* tersebut dikonversi menjadi suatu titik koordinat x dan y. Misal n-nya adalah 9 maka ukuran papan adalah 9×9 dan suatu *block* didefinisikan sebagai titik koordinat, misalnya (x,y) = (3,1).

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)	(7,0)	(8,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)	(8,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)	(7,2)	(8,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	(7,3)	(8,3)
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Gambar 1. Visualisasi block as coordinate

2. **Aturan baris**

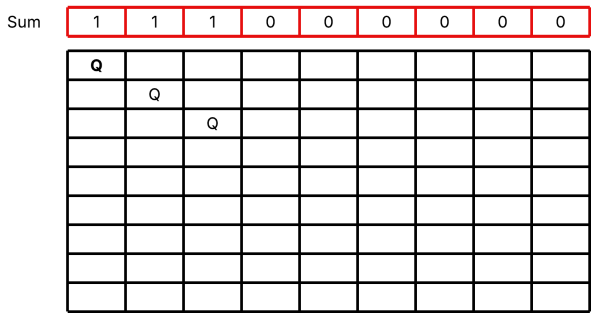
   Melakukan *count* pada suatu baris di mana maksimal *count* pada suatu baris adalah 1 dan baris yang kosong adalah 0. Jika count tersebut sudah 1 berarti pada *block* tersebut sudah ditempati *queen* lain.



Gambar 2. Visualisasi aturan baris

3. **Aturan kolom**

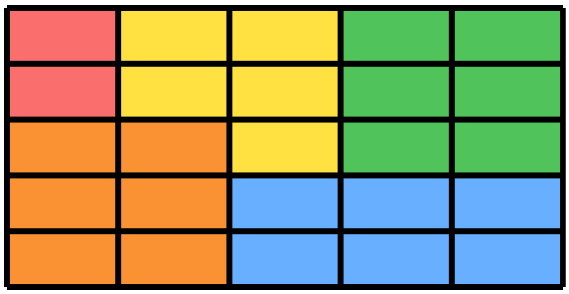
   Sama seperti halnya aturan baris, aturan kolom juga menggunakan pendekatan *count* pada suatu kolom. *Count* bernilai 0 jika pada suatu kolom tidak ada queen yang ditempatkan. Jika terdapat *count* 1 pada suatu kolom, maka kolom tersebut sudah ditempatkan *queen.*



Gambar 3. Visualisasi aturan kolom

4. **Aturan warna**

   Aturan warna ini akan mengubah suatu kode huruf pada input .txt itu menjadi kumpulan koordinat. Misal kode huruf A itu lengkap pada baris pertama. Maka A = [(1,1), (2,1), (3,1), ...]. Ini digunakan sebagai *constraint* pada fungsi isValid untuk pengecekan pada penempatan *queen.*



Gambar 4. Visualisasi aturan warna

Misalnya n adalah 5 sehingga 5 x5. Maka dengan data berikut

```
data_file = [
ABBCC
ABBCC
DDBCC
DDEEE
DDEEE
]
```

Akan menghasilkan data warna sebagai berikut.

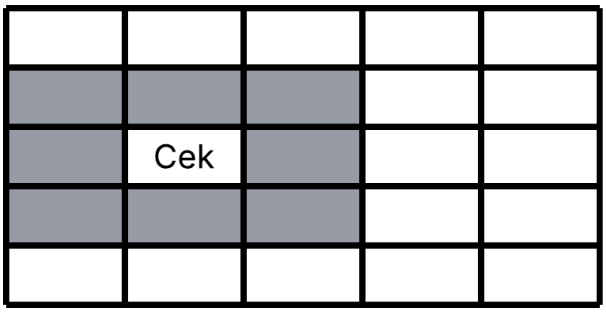
```
colored_area = [
    [(0, 0), (1, 0)],
    [(0, 1), (0, 2), (1, 1), (1, 2), (2, 2)],
    [(0, 3), (0, 4), (1, 3), (1, 4), (2, 3), (2, 4)],
    [(2, 0), (2, 1), (3, 0), (3, 1), (4, 0), (4, 1)],
```

```
    [(3, 2), (3, 3), (3, 4), (4, 2), (4, 3), (4, 4)]
]
```

Yang mana baris pertama adalah area A, baris kedua area B, baris ketiga area C, dan seterusnya. Jadi setiap baris adalah kumpulan dari titik-titik dengan daerah yang sama.

5. **Aturan tetangga**

   Aturan tetangga ini akan mengecek tetangga suatu *block* baik tetangga samping, atas, bawah, serta diagonalnya untuk memastikan penempatan tidak bersinggungan dengan aturan tetangga pada definisi masalah.



Gambar 5. Visualisasi aturan tetangga

STRUKTUR DATA

1. **Struktur data *board***
   List of Lists (berarti list 2 dimensi) berisikan Integer dengan representasi 0 untuk kotak kosong dan 1 untuk kondisi terdapat queen.

Stuktur data

`board = [`
`  [0, 1, 0, 0],`
`  [0, 0, 0, 1],`
`  [1, 0, 0, 0],`
`  [0, 0, 1, 0]`
`]`

2. **Struktur data untuk area warna (*colored area*)**
   List of Lists of Tuples, merupakan sebuah list yang mana setiap elemennya adalah list koordinat millik suatu warna tertentu. Awalnya saya membuat dictionary agar mudah dikelompokkan per huruf, lalu diubah jadi List of Values agar mudah diiterasi.

Pemisalan peta warna	**Struktur datanya**

AA
BB
	`colored_area = [`
	`  [(0, 0), (0, 1)],`
	`  [(1, 0), (1, 1)]`
	`]`

   Jadi saat validasi, program akan looping list ini. Jika ratu disimpan di (0,0), program akan mengecek apakah di (0,1) sudah ada ratu atau belum

3. **Data mentah (data file/raw_lines) –** *default*
   Representasi peta warna dalam teks asli dengan tipe data List of Strings, setiap elemen list adalah 1 baris string dari file txt.

Stuktur data

`data_file = [`
`  "AABB",`
`  "AACC",`
`  "DDCC",`
`  "DDEE"`
`]`

PENJELASAN FUNGSI

Fungsi	Penjelasan
`initial_board()`	Membuat papan permainan kosong dengan cara membuat list 2 dimensi berukuran n x n yang seluruh isinya diinisiasi dengan angka 0
`is_valid(board, row, col, colored_area)`	Memeriksa penempatan ratu di suatu posisi yakni (row, col) apakah melanggar aturan atau tidak. Fungsi akan memeriksa baris dan kolom untuk memastikan tidak ada angka 1 lain di baris atau kolom yang sama. Untuk diagonal, fungsi memastikan tidak ada angka 1 di kotak yang bersentuhan secara diagonal (jaraknya 1 langkah) dan pengecekan warna untuk mencari area warna dari koordinat tersebut, lalu memastikan di dalam areanya hanya terdapat maksimal 1 ratu.
`data_color(area)`	Mengubah teks peta warna menjadi data koordinat dengan cara membaca input string per baris. Jika menemukan karakter misaknya A, ia akan menyimpan koordinat (baris, kolom) ke dictionary yang dikelompokkan berdasarkan karakternya.
`load_file(filename)`	Membaca file txt dan menampilkannyya ke terminal dengan ANSI color.
`run_brute_force(board, row, area, raw_lines)`	Algoritma untuk mencari solusi dengan mengisi papan demi baris dengan queen. Jika papan sudah penuh, ia baru memeriksa validitas seluruh papan menggunakan is_valid. Jika valid, akan mengembalikan True, jika tidak akan melakukan pengubahan dari 1 jadi 0 lagi dan mencoba posisi lain.
`print_board(board, raw_lines, durasi, jumlah_kasus)`	Menampilkan hasil akhir solsi ke terminal dan menyimpan ke file dengan cara kerja menggabungkan data papan dengan huruf asli (raw_lines). Jika ada ratu dicetak dengan # dan kosong cetak huruf aslinya. Selain itu juga akan menampilkan waktu dan jumlah kasusnya.
`print_live(board, n, raw_lines)`	Melakukan penampilkan board secara live update. Mengecek waktu sekarang. Jika selisih dengan waktu print terakhir < 0.3 s, maka akan berhenti. Lalu membersihkan layar terminal, dan mencetak ulang papan sekarang.

LANGKAH ALGORITMA *BRUTE FORCE*

Seperti yang dijelaskan dalam QnA bahwa backtracking tanpa early pruning diperbolehkan sehingga saya melakukan pengisian queen hingga baris bawah sampai papan penuh lalu melakukan pengecekan. Sebab saya melakukan pengecekan validitas setelah papan terisi penuh, algoritma yang dibuat akan membuang waktu menyusun ribuan kombinasi di baris bawah, padahal kesalahan ada di baris paling atas misalnya. Oleh karena itu, algoritma ini saya susun dengan konsep tersebut.

Kode

```
def run_brute_force(board, row, area, raw_lines=None):
    global stats
    n = len(board)

    if raw_lines is not None:
        print_live(board, n, raw_lines)

    if row == n:
        stats += 1
        for r in range(n):
            c = -1
            for k in range(n):
                if board[r][k] == 1:
                    c = k
                    break
            if c == -1: return False
            if is_valid(board, r, c, area)== False:
                return False
        return True

    for col in range(n):
        board[row][col] = 1
        if run_brute_force(board, row + 1, area, raw_lines):
            return True
        board[row][col] = 0
    return False
```

Untk base case saya menggunakan pengondisian yang mana suatu kasus hanya akan dicek ketika row == n atau ketika papan sudah penuh.

Algoritma akan mencari koordinat titik yang akan dicek untuk setiap baris dan kolom pada perulangan. Sebagai fungsi pengecekan saya menggunakan fungsi is_valid yang akan melakukan pengecekan beberapa syarat berikut.

1. Pengecekan baris dan kolom
2. Pengecekan tetanggan dengan jarak 1 secara diagonal

3. Pengecekan warna.

Lalu setelahnya akan return boolean untuk validitas penempatan queen.

Setelah itu ada konsep rekursif yang dijalankan saat papan belum penuh (row < n). Pada kode ini, tidak ada pengecekan validitas, sehingga sedikit 'memaksa' lanjut walaupun ratu yang baru ditaruh itu tabrakan. Namun jika menemukan solusi, pengkondisian run_brute_force akan me-*return* True bahwa algoritma sudah menemukan solusi. Jika tidak, maka melakukan backtracking pada kolom berikutnya. Jika pada seluruh kemungkinan tidak mendapatkan solusi, maka akan dikembalikan sebuah boolean False.

Selain fungsi brute force utama yang penting, terdapat fungsi is_valid juga yang tidak kalah pentingnya untuk melakukan pengecekan pada aturan-aturan yang ada.

Kode

```
def is_valid(board, row, col, colored_area):
    n = 0
    for x in board:
        n = n + 1

    # Baris dan kolom
    for i in range(n):
        if i != row and board[i][col] == 1:
            return False
        if i != col and board[row][i] == 1:
            return False

    # Diagonal
    for r in range(n):
        for c in range(n):
            if board[r][c] == 1:
                if r != row or c != col:
                    jarak_baris = row - r
                    if jarak_baris < 0:
                        jarak_baris = -jarak_baris
                    jarak_kolom = col - c
                    if jarak_kolom < 0:
                        jarak_kolom = -jarak_kolom
                    if jarak_baris == 1 and jarak_kolom == 1:
                        return False

    # Area berwarna
    my_area = None
    for area in colored_area:
        for pos in area:
            if pos[0] == row and pos[1] == col:
                my_area = area
                break
        if my_area: break

    if my_area:
```

```
        count = 0
        for pos in my_area:
            if board[pos[0]][pos[1]] == 1:
                count += 1
        if count > 1:
            return False

    return True
```

Fungsi ini melakukan validasi pada papan, di mana pertama-tama akan menghitung ukuran papan.

Cek baris dan kolom
Melakukan pengecekan apakah ada ratu lain di kolom dan baris yang sama, lalu jika ada akan return False.

Cek diagonal
Melakukan pengecekan jika terdapat ratu lain di papan, tapi bukan ratu yang sedang dicek, akan dihitung jarak baris dan kolomnya (selisih absolut), lalu memastikan bahwa tidak boleh bersentuhan secara diagonal (jarak 1 langkah).

Cek aturan warna
Mencari tahu (row,col) ini termasuk area warna yang mana. Lalu menghitung jumlah ratu di dalam area tersebut. Jika terdapat lebih dari 1 ratu, berarti melanggar aturan.

KODE PROGRAM

Sebagai catatan, beberapa fungsi pada utils.py tidak digunakan pada implementasi GUI-nya karena pada awalnya digunakan untuk melakukan testing pada algoritma di CLI karena dibutuhkan penyesuaian pada library terkait misalnya fungsi load_file akan dibuat 'versi' library tkinter-nya di file interface.py. Namun, logika yang ditulis di utils.py menjadi *guide* untuk menuliskan 'versi' library GUI-nya. Saya memutuskan untuk tidak menghapus file main.py dan beberapa fungsi di utils sebagai arsip.

utils.py

```
import time
import os
from PIL import Image, ImageDraw, ImageFont
stats = 0
last_print_time = 0

---

def initial_board(n):
    board = [[0 for i in range(n)] for j in range(n)]
    return board

---

def is_valid(board, row, col, colored_area):
    n = 0
    for x in board:
        n = n + 1

    # Baris dan kolom
    for i in range(n):
        if i != row and board[i][col] == 1:
            return False
        if i != col and board[row][i] == 1:
            return False

    # Diagonal
    for r in range(n):
        for c in range(n):
            if board[r][c] == 1:
                if r != row or c != col:
                    jarak_baris = row - r
                    if jarak_baris < 0:
                        jarak_baris = -jarak_baris
                    jarak_kolom = col - c
                    if jarak_kolom < 0:
                        jarak_kolom = -jarak_kolom
                    if jarak_baris == 1 and jarak_kolom == 1:
                        return False

    # Area berwarna
    my_area = None
```

```
    for area in colored_area:
        for pos in area:
            if pos[0] == row and pos[1] == col:
                my_area = area
                break
        if my_area: break

    if my_area:
        count = 0
        for pos in my_area:
            if board[pos[0]][pos[1]] == 1:
                count += 1
        if count > 1:
            return False

    return True
```

```
def data_color(area):
    rows = area.strip().split('\n')
    n = len(rows)
    areas = {}

    for i in range(n):
        rc = rows[i].strip() # rc = color in row i

        if not rc: continue

        for j in range(len(rc)):
            color = rc[j]
            if color.isalnum():
                if color not in areas:
                    areas[color] = []
                areas[color].append((i,j))

    return list(areas.values())
```

```
def load_file(filename):
    with open(f"../test/input/{filename}.txt", 'r') as f:
        content = f.read()

    colors = [
        "\033[91m", "\033[92m",
        "\033[93m", "\033[94m",
        "\033[95m", "\033[96m",
        "\033[31m", "\033[32m",
        "\033[33m", "\033[34m",
        "\033[35m", "\033[36m",
        "\033[91;1m", "\033[92;1m",
        "\033[93;1m", "\033[94;1m",
        "\033[95;1m", "\033[96;1m",
        "\033[37m", "\033[90m",
        "\033[41m", "\033[42m",
        "\033[43m", "\033[44m",
        "\033[45m", "\033[46m"
    ]
    reset = "\033[0m"
```

```
    unique_chars = []

    for i in range(len(content)):
        char = content[i]

        if char == '\n':
            continue
        if char == ' ':
            continue

        exist = False
        for j in range(len(unique_chars)):
            if unique_chars[j] == char:
                exist = True
                break

        if exist == False:
            unique_chars.append(char)

        r = ""
        for i in range(len(content)):
            char = content[i]
            if char == '\n' or char == ' ':
                r += char
                continue

            idx = 0
            for j in range(len(unique_chars)):
                if unique_chars[j] == char:
                    idx = j
                    break
            while idx >= len(colors):
                idx -= len(colors)

            selected = colors[idx]

            r += selected + char + reset
    return r
```

```
def run_brute_force(board, row, area, raw_lines=None):
    global stats
    n = len(board)

    if raw_lines is not None:
        print_live(board, n, raw_lines)

    if row == n:
        stats += 1
        for r in range(n):
            c = -1
            for k in range(n):
                if board[r][k] == 1:
                    c = k
                    break
            if c == -1: return False
            if is_valid(board, r, c, area)== False:
                return False
        return True
```

```
    for col in range(n):
        board[row][col] = 1
        if run_brute_force(board, row + 1, area, raw_lines):
            return True
        board[row][col] = 0
    return False
```

```
def print_board(board, raw_lines, durasi, jumlah_kasus):
    n = len(board)
    txt_res = ""

    for r in range(n):
        row_str = ""

        if r < len(raw_lines):
            str_original = raw_lines[r].strip()
        else:
            str_original = ""*n

        for c in range(n):
            if board[r][c] == 1:
                row_str = row_str + '#'
            else:
                if c < len(str_original):
                    row_str += str_original[c]
                else:
                    row_str += "."

        txt_res += row_str + "\n"

    waktu_pencarian = "Waktu pencarian: " + str(int(durasi * 1000)) + " ms\n"
    banyak_kasus = "Jumlah kasus yang ditinjau: " + str(jumlah_kasus) + "\n"

    result = txt_res + "\n"+ waktu_pencarian + banyak_kasus

    print(result)

    save = input("Apakah Anda ingin menyimpan hasilnya? (y/n): ")
    if save == 'y' or save == 'Y' or save == 'ya' or save == 'Ya':
        filename = input("Masukkan nama file untuk menyimpan hasil: ")
        with open(f"../test/output/{filename}.txt", 'w') as f:
            f.write(result)
        print("Berhasil disimpan")
    else:
        print("Hasil tidak disimpan")
```

```
def print_live(board, n, raw_lines):
    global last_print_time

    current_time = time.time()

    if current_time - last_print_time < 0.3:
        return

    last_print_time = current_time

    os.system('cls' if os.name == 'nt' else 'clear')
```

```
    txt_res = ""
    for r in range(n):
        row_str = ""
        if r < len(raw_lines):
            str_original = raw_lines[r].strip()
        else:
            str_original = ""*n

        for c in range(n):
            if board[r][c] == 1:
                row_str = row_str + '# '
            else:
                if c < len(str_original):
                    char = str_original[c]
                else:
                    char = "."
                row_str += char + " "
        txt_res += row_str + "\n"

    print(txt_res)
    print("coba kombinasi")
```

```
def save_image(board, raw_lines, filename, stats, duration):
    n = len(board)
    cell_size = 60
    margin_bot = 100
    width = n * cell_size
    height = n * cell_size + margin_bot

    img = Image.new('RGB', (width, height), color='white')
    draw = ImageDraw.Draw(img)

    color_palette = [
        (255, 179, 186), (255, 223, 186), (255, 255, 186), (186, 255, 201), (186,
225, 255),
        (230, 179, 255), (255, 179, 230), (212, 240, 240), (255, 198, 255), (189,
178, 255),
        (160, 196, 255), (155, 246, 255), (202, 255, 191), (253, 255, 182), (255,
214, 165),
        (255, 173, 173), (200, 200, 200), (240, 230, 140), (176, 224, 230), (144,
238, 144)
    ]

    for r in range(n):
        if r < len(raw_lines):
            str_original = raw_lines[r].strip()
        else:
            str_original = ""*n

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

            if c < len(str_original):
                char = str_original[c]
                color_idx = ord(char)%len(color_palette)
```

```
                fill_color = color_palette[color_idx]
            else:
                fill_color = (255, 255, 255)

            draw.rectangle([x0, y0, x1, y1], fill=fill_color, outline='black',
width=2)

            if board[r][c] == 1:
                padding = 10
                draw.ellipse([x0 + padding, y0 + padding, x1 - padding, y1 -
padding], fill='black')

                text_x = x0 + (cell_size // 3)
                text_y = y0 + (cell_size // 4)
                draw.text((text_x, text_y), 'Q', fill='white')
    text_x = 20
    text_y = (n * cell_size) + 20

    text_info = f"Waktu: {int(duration * 1000)} ms\nKasus ditinjau: {stats}"

    draw.text((text_x, text_y), text_info, fill="black")

    img.save(f"../test/output/image/{filename}.png")
```

main.py

```
from utils import *
import utils
from time import time

def main():
    n = int(input("Masukkan n: "))
    filename = input("Masukkan nama file: ")
    with open(f"../test/input/{filename}.txt", 'r') as f:
        raw_content = f.read()

    raw_lines = raw_content.strip().split('\n')

    if n != len(raw_lines):
        print("Input tidak sesuai dengan n, silakan cek lagi input")
        return

    vis_input = load_file(filename)
    print(vis_input)
    print("\n")

    colored_area = data_color(raw_content)

    board = initial_board(n)
    utils.stats = 0

    start_time = time()
    result = run_brute_force(board, 0, colored_area, raw_lines)
    print_live(board, n, raw_lines)

    end_time = time()
    durasi = end_time - start_time

    if result:
        print_board(board, raw_lines, durasi, utils.stats)
```

```
    else:
        print("Tidak ada solusi yang ditemukan.")
        print("Waktu pencarian: " + str(int(durasi * 1000)) + " ms")
        print("Jumlah kasus yang ditinjau: " + str(utils.stats))

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

ANALISIS KOMPLEKSITAS

Time Complexity

Merupakan cara kita dalam mengukur seberapa cepat *workload* algoritma saat jumlah data atau n diperbesar. *Time complexity* ini bukan mengukur waktu dalam detik atau milidetik, melainkan mengukur jumlah langkah operasi yang harus dilakukan komputer.

Algoritma yang dibuat memiliki rekursif di mana setiap baris mencoba mengisi setiap kolom dari 0 sampai n-1. Karena tidak melakukan *pruning*, algoritma akan terus bercabang sampai kedalaman n.

$$N \times N \times N \ldots (sebanyak\ N\ kali) = N^N$$

Setiap kali algoritma mencapai dasar, ia menjalankan fungsi validasi. Di dalamnya, fungsi is_valid dipanggil di mana fungsi ini memiliki *nested loop* untuk mengecek diagonal dan area warna, yang kompleksitasnya sekitar $O(N^2)$. Sehingga total kompleksitas waktunya $O(N^N \times N^3)$ atau $O(N^N)$.

Space Complexity

Merupakan cara untuk mengukur berapa banyak memori (RAM) yang dibutuhkan oleh algoritma untuk menyelesaikan tugasnya seiring dengan bertambahnya jumlah data (n).

Secara sederhana dalam menyimpan papan dalam matriks n x n.

$$O(N^2)$$

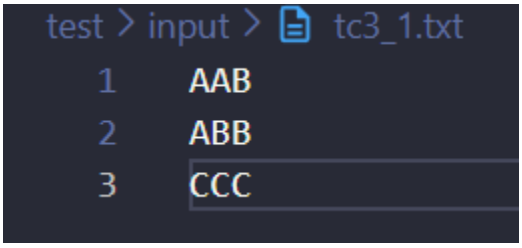
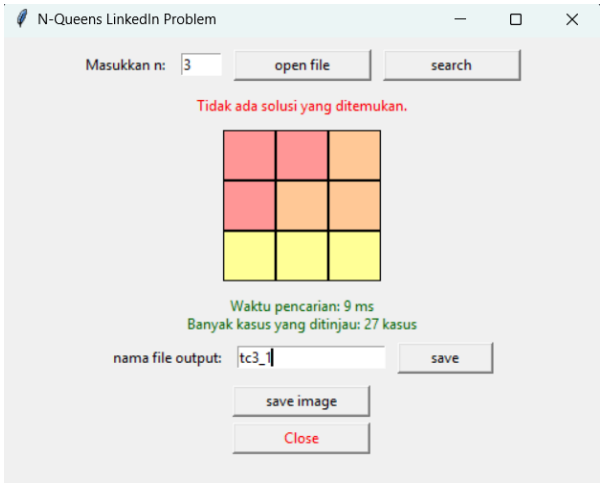
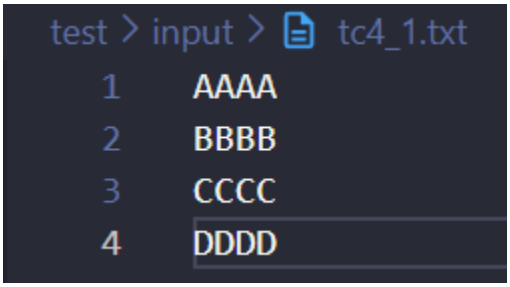
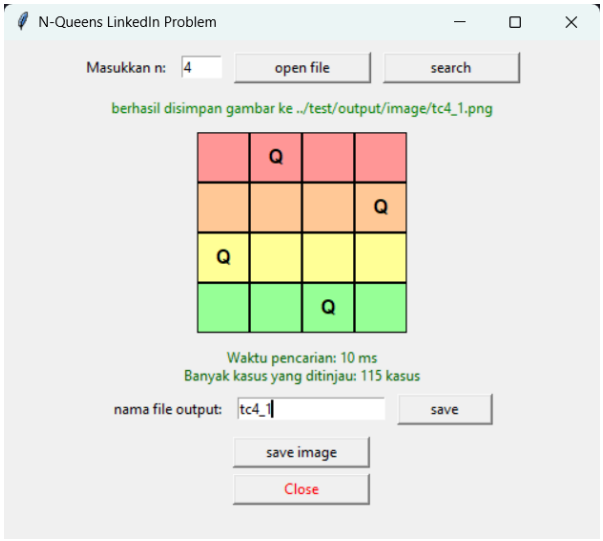
Selanjutnya rekursi akan berjalan sedalam jumlah baris (N). Setiap pemanggilan rekursi akan menyimpan *variabl*e lagi.

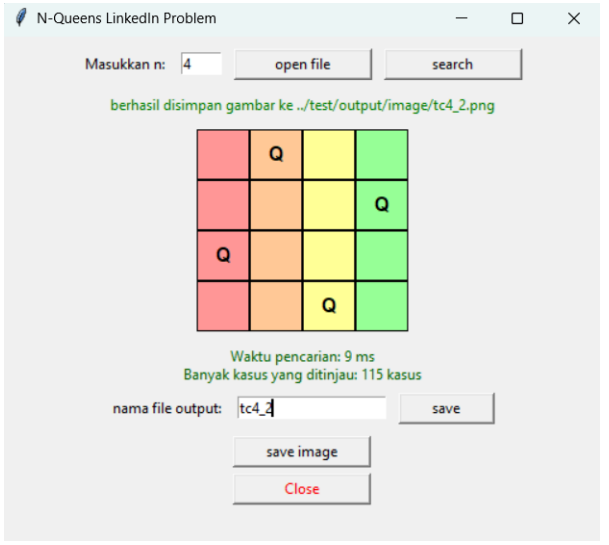
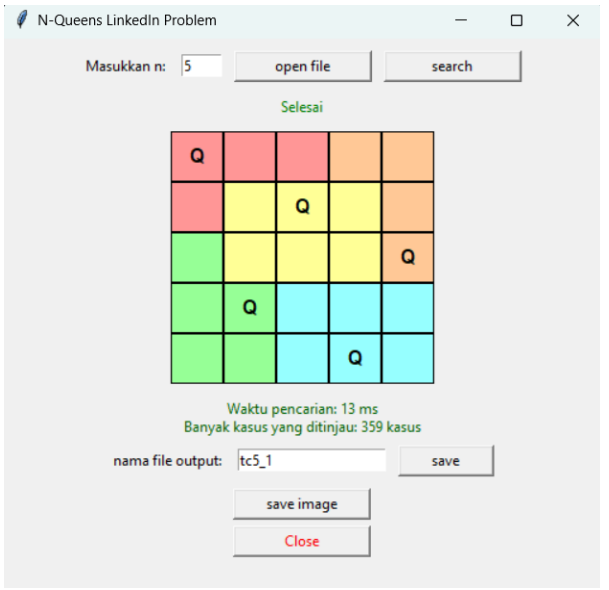
$$O(N)$$

Sehingga total kompleksitas ruangnya adalah $O(N^2)$.

PENGUJIAN

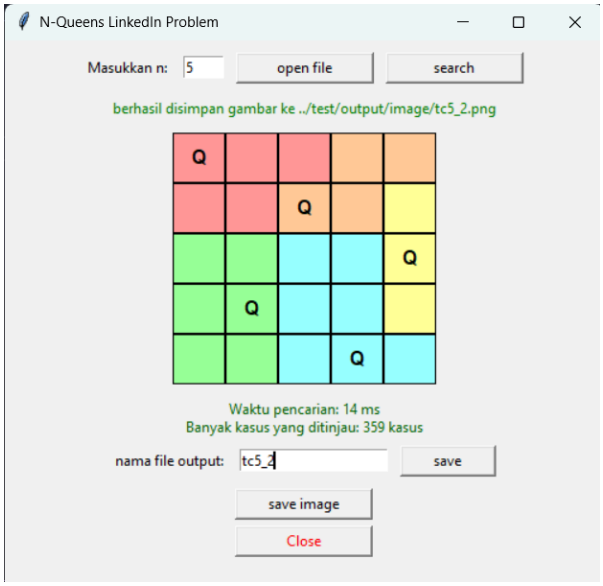
Pengujian *Test Case*

Input	Output
	
	

<table>
<tr>
<td>test > input > tc4_2.txt
1 ABCD
2 ABCD
3 ABCD
4 ABCD</td>
<td> 
</td>
</tr>
<tr>
<td>test > input > tc5_1.txt
1 AAABB
2 ACCCB
3 DCCCB
4 DDEEE
5 DDEEE</td>
<td> 
</td>
</tr>
</table>

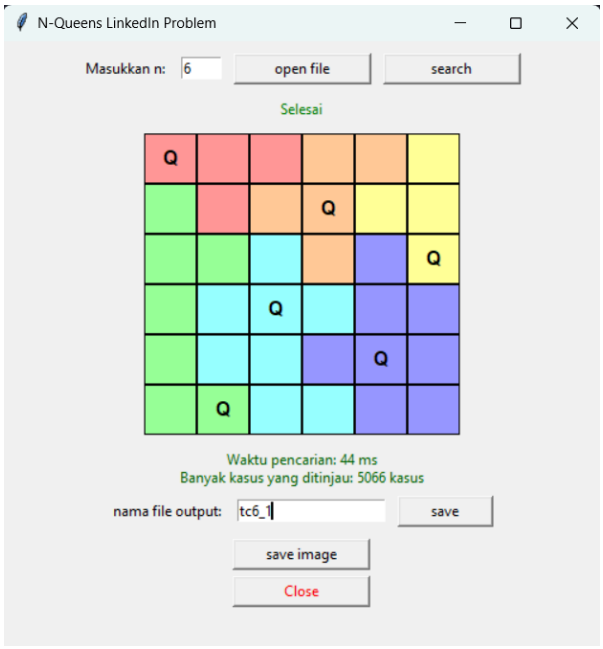
test > input > tc5_2.txt

```
1  AAABB
2  AABBC
3  DDEEC
4  DDEEC
5  DDEEE
```



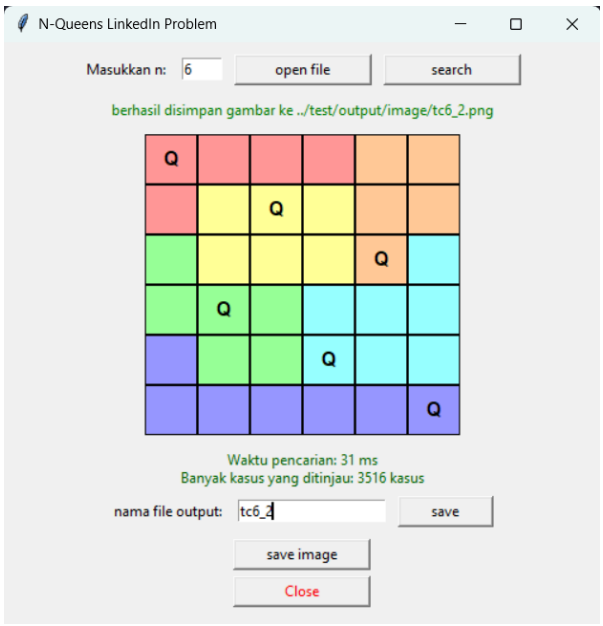
test > input > tc6_1.txt

```
1  AAABBC
2  DABBCC
3  DDEBFC
4  DEEEFF
5  DEEFFF
6  DDEEFF
```



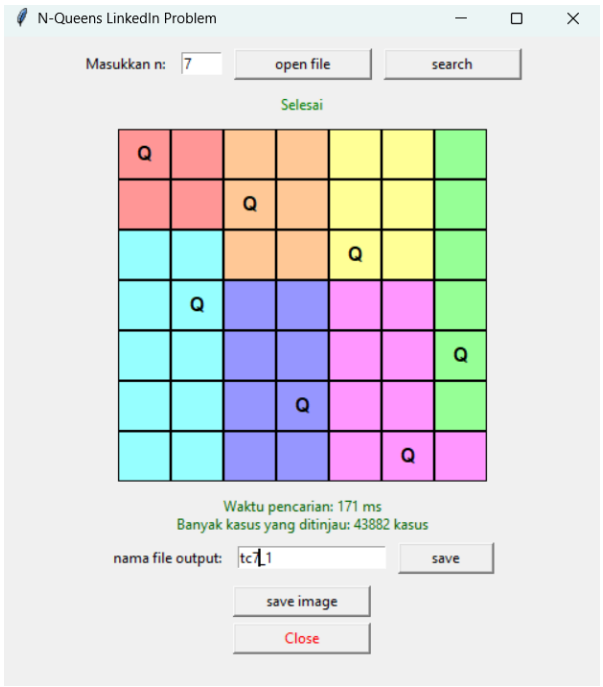
test > input > tc6_2.txt

```
1 AAAABB
2 ACCCBB
3 DCCCBE
4 DDDEEE
5 FDDEEE
6 FFFFFF
```



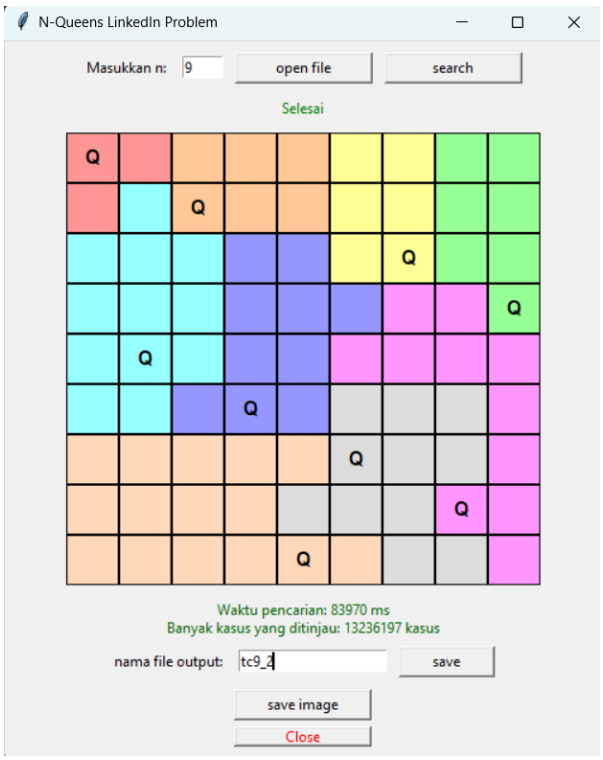
test > input > tc7_1.txt

```
1 AABBCCD
2 AABBCCD
3 EEBBCCD
4 EEFFGGD
5 EEFFGGD
6 EEFFGGD
7 EEFFGGG
```



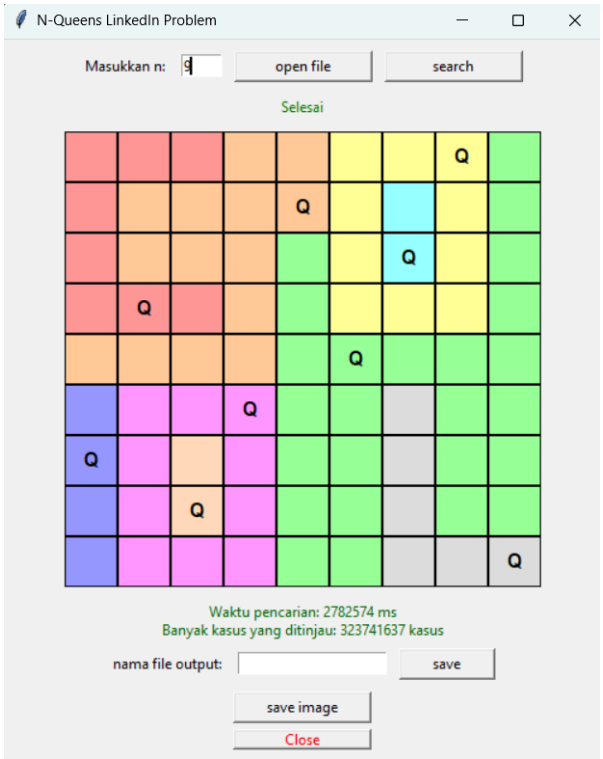
test > input > tc9_2.txt

```
1  AABBBCCDD
2  AEBBBCCDD
3  EEEFFCCDD
4  EEEFFFGGD
5  EEEFFGGGG
6  EEFFFHHHG
7  IIIIIHHHG
8  IIIIHHHGG
9  IIIIIIHHG
```

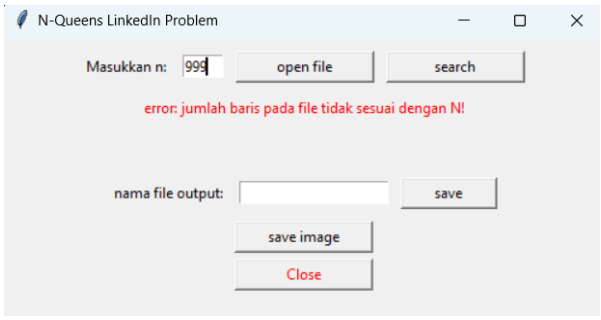
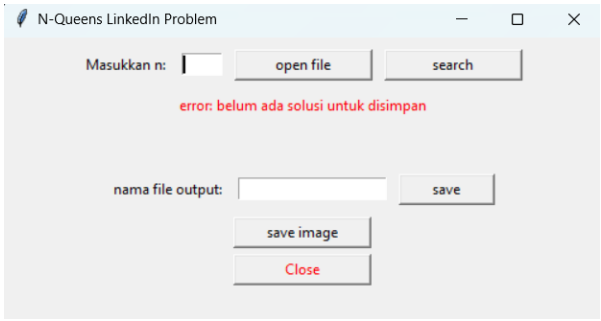
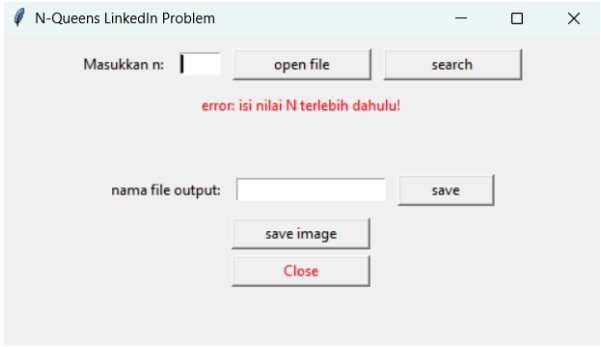
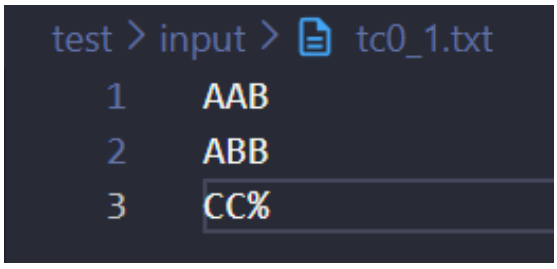
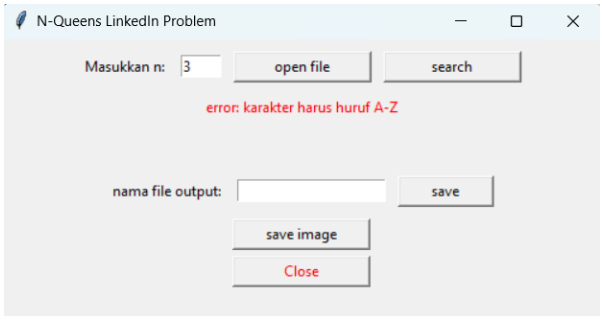


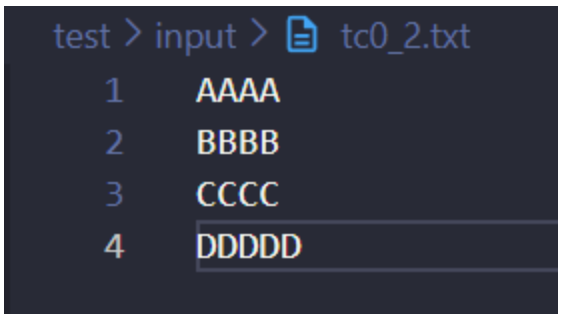
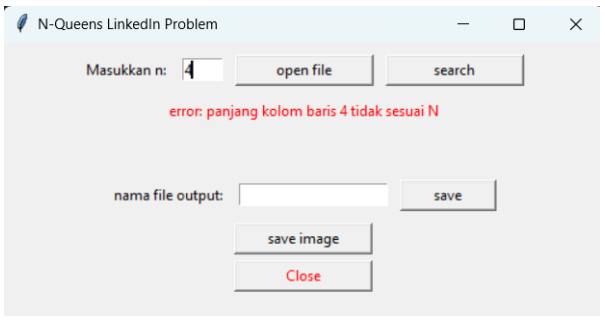
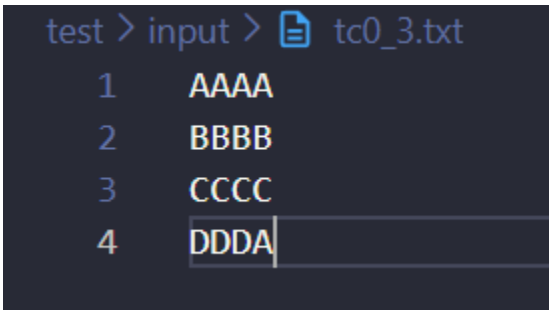
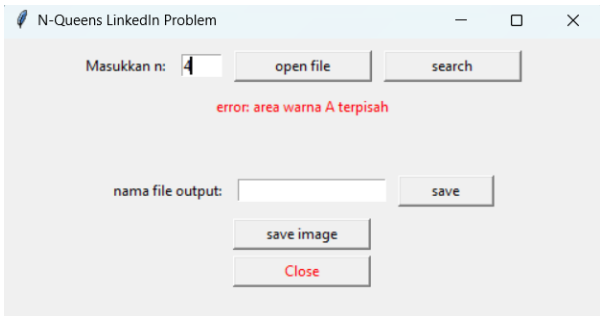
test > input > tc9_1.txt

```
1  AAABBCCCD
2  ABBBBCECD
3  ABBBDCECD
4  AAABDCCCD
5  BBBBDDDDD
6  FGGGDDHDD
7  FGIGDDHDD
8  FGIGDDHDD
9  FGGGDDHHH
```



Pengujian Validitas

Input	Output
Input n = 999 dan tidak sesuai dengan board yang dimuat.	
Tidak bisa melakukan save sebelum solusi ditemukan. Jika tidak menghasilkan solusi, tidak dapat melakukan save baik berupa txt maupun image.	
Jika membuka file, namun N belum diisi.	
	

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
</table>

SPESIFIKASI BONUS

Graphical User Interface

Pada kesempatan ini saya juga mencoba mengerjakan GUI untuk memvisualisasikan algoritma brute force yang sudah dibuat. Alasan saya menggunakan tkinter sebagai tools dalam membuat visualisasi ini karena pada tugas besar IF3070 Dasar Inteligensi Artifisial saya pernah menggunakan tkinter ini sehingga lebih familiar. Selain itu saya terinspirasi oleh permainan catur yang menggunakan library ini pada platform YouTube(https://youtu.be/VgVHFQOeAng?si=eVrk1KYYYprSRnd8).

File ini bertindak sebagai frontend/controller. Ia menangani input user, validasi data, visualisasi, dan interaksi, sementara logika "berpikir" (algoritma brute force) diserahkan ke modul utils. Beberapa fungsi pada file ini disesuaikan dengan cara kerja library tkinter, namun tetap dalam pendekatan logika pada fungsi-fungsi di utils.py.

Secara sederhana flow saya menggunakan library ini adalah dengan mengidentifikasi terlebih dahulu *variables* yang diperlukan. Lalu membuat button atau frame yang dibutuhkan dengan logika yang digunakan bisa memanggil sebuah fungsi pada utils.py atau membuat logika terpisah. Beberapa hal seperti bug pada *clear frame* dapat diselesaikan dengan bantuan referensi seperti referensi berikut https://stackoverflow.com/questions/15781802/python-tkinter-clearing-a-frame.

interface.py

```
import tkinter as tk
from tkinter import filedialog
import utils
import time

---

class QueenLinkedIn:
    def __init__(self):
        self.window = tk.Tk()
        self.window.title("N-Queens LinkedIn Problem")
        self.window.geometry("500x600")

        self.n = 0
        self.data_file = []
        self.color_area = []
        self.block = []
        self.last_print_time = 0
        self.durasi = 0
        self.solution_board = []

        self.frame_top = tk.Frame(self.window)
```

```
        self.frame_top.pack(pady=10)

        self.label_n = tk.Label(self.frame_top, text="Masukkan n:")
        self.label_n.pack(side=tk.LEFT, padx=5)

        self.input_n = tk.Entry(self.frame_top, width=5)
        self.input_n.pack(side=tk.LEFT, padx=5)

        self.btn_open = tk.Button(self.frame_top, text="open file",
command=self.open_file, width=15)
        self.btn_open.pack(side=tk.LEFT, padx=5)

        self.btn_search = tk.Button(self.frame_top, text="search",
command=self.search, width=15)
        self.btn_search.pack(side=tk.LEFT, padx=5)

        self.label_status = tk.Label(self.window, text="masukkan N lalu pilih file
input.")
        self.label_status.pack()

        self.board_frame = tk.Frame(self.window)
        self.board_frame.pack(pady=10)

        self.frame_bot = tk.Frame(self.window)
        self.frame_bot.pack()

        self.label_hasil = tk.Label(self.frame_bot, text="", fg="darkgreen")
        self.label_hasil.pack()

        self.frame_save = tk.Frame(self.window)
        self.frame_save.pack(pady=5)

        self.label_save = tk.Label(self.frame_save, text="nama file output:")
        self.label_save.pack(side=tk.LEFT, padx=5)

        self.input_filename = tk.Entry(self.frame_save, width=20)
        self.input_filename.pack(side=tk.LEFT, padx=5)
        self.input_filename.insert(0, "")

        self.btn_save = tk.Button(self.frame_save, text="save",
command=self.save_result, width=10)
        self.btn_save.pack(side=tk.LEFT, padx=5)

        self.frame_save_img = tk.Frame(self.window)
        self.frame_save_img.pack(pady=5)

        self.btn_save_img = tk.Button(self.frame_save_img, text="save image",
command=self.save_image, width=15)
        self.btn_save_img.pack()

        self.btn_close = tk.Button(self.frame_save_img, text="Close",
command=self.window.destroy, width=15, fg="red")
        self.btn_close.pack(pady=5)

        self.window.mainloop()
```

```
    def open_file(self):
        isi_n = self.input_n.get()
        if isi_n == "":
```

```
            self.label_status.config(text="error: isi nilai N terlebih dahulu!",
fg="red")
            return

        if isi_n.isdigit() == False:
            self.label_status.config(text="error: nilai N harus berupa angka!",
fg="red")
            return

        self.n = int(isi_n)

        filename = tk.filedialog.askopenfilename(initialdir="../test/input")
        if filename == "":
            return

        f = open(filename, 'r')
        isi_text = f.read()
        f.close()

        raw_lines = isi_text.split('\n')
        temp_data = []
        for line in raw_lines:
            clean = line.strip()
            if len(clean) > 0:
                temp_data.append(clean)

        if len(temp_data) != self.n:
            self.label_status.config(text="error: jumlah baris pada file tidak
sesuai dengan N!", fg="red")
            for widget in self.board_frame.winfo_children():
                widget.destroy()
            self.data_file = []
            return

        color_groups = {}

        for r in range(self.n):
            row_str = temp_data[r]
            if len(row_str) != self.n:
                self.label_status.config(text="error: panjang kolom baris " +
str(r+1) + " tidak sesuai N", fg="red")
                for widget in self.board_frame.winfo_children():
                    widget.destroy()
                self.data_file = []
                return

            for c in range(self.n):
                char = row_str[c]
                if char.isalpha() == False:
                    self.label_status.config(text="error: karakter harus huruf A-Z",
fg="red")
                    for widget in self.board_frame.winfo_children():
                        widget.destroy()
                    self.data_file = []
                    return

                if char not in color_groups:
                    color_groups[char] = []
                color_groups[char].append((r, c))

        if len(color_groups) != self.n:
```

```
            self.label_status.config(text="error: jumlah warna harus sama dengan N",
fg="red")
            for widget in self.board_frame.winfo_children():
                widget.destroy()
            self.data_file = []
            return

        for char in color_groups:
            coords = color_groups[char]
            if len(coords) == 0:
                continue

            start_node = coords[0]
            queue = [start_node]
            visited = []
            visited.append(start_node)
            count = 0

            while len(queue) > 0:
                curr = queue.pop(0)
                r = curr[0]
                c = curr[1]
                count += 1

                neighbors = [[-1, 0], [1, 0], [0, -1], [0, 1]]
                for move in neighbors:
                    nr = r + move[0]
                    nc = c + move[1]

                    if (nr, nc) in coords:
                        is_visited = False
                        for v in visited:
                            if v == (nr, nc):
                                is_visited = True
                                break

                        if is_visited == False:
                            visited.append((nr, nc))
                            queue.append((nr, nc))

            if count != len(coords):
                self.label_status.config(text="error: area warna " + char + "
terpisah", fg="red")
                for widget in self.board_frame.winfo_children():
                    widget.destroy()
                self.data_file = []
                return

        self.color_area = utils.data_color(isi_text)
        self.data_file = temp_data
        self.solution_board = []

        self.board_ui()
        self.label_status.config(text="file valid", fg="black")
        self.label_hasil.config(text="")

    def board_ui(self):
        for widget in self.board_frame.winfo_children():
            widget.destroy()

        self.block = []
```

```
        color_list = [
            "#FF9999", "#FFCC99", "#FFFF99", "#99FF99", "#99FFFF", "#9999FF",
"#FF99FF", "#E0E0E0",
            "#FFDAB9", "#E6E6FA", "#F0E68C", "#ADD8E6", "#98FB98", "#FFB6C1",
"#D3D3D3", "#F5DEB3",
            "#87CEEB", "#FFA07A", "#EE82EE", "#40E0D0", "#F08080", "#9ACD32",
"#DA70D6", "#B0E0E6",
            "#FFEFD5", "#D2B48C"
        ]

        for i in range(self.n):
            block_row = []
            txt_row = self.data_file[i].strip()
            for j in range(self.n):
                huruf = txt_row[j]

                if huruf.isalpha():
                    color_idx = (ord(huruf.upper()) - 65) % 26
                else:
                    color_idx = 0

                color_bg = color_list[color_idx]

                block_box = tk.Label(self.board_frame, text="", width=4, height=2,
bg=color_bg, borderwidth=1, relief="solid", font=("Arial", 12, "bold"))

                block_box.grid(row=i, column=j)
                block_row.append(block_box)
            self.block.append(block_row)
```

```
    def update_screen(self, board, n, raw_lines):
        current_time = time.time()
        if current_time - self.last_print_time < 0.2:
            return

        self.last_print_time = current_time
        for i in range(n):
            for j in range(n):
                if board[i][j] == 1:
                    self.block[i][j].config(text="Q", fg="black")
                else:
                    self.block[i][j].config(text="")
        self.label_status.config(text="mencari solusi... langkah ke: " +
str(utils.stats), fg="blue")
        self.window.update()
```

```
    def search(self):
        if self.n == 0:
            self.label_status.config(text="error: silakan buka file terlebih
dahulu", fg="red")
            return

        if len(self.data_file) == 0:
            self.label_status.config(text="error: file kosong atau tidak valid",
fg="red")
            return
```

```
        utils.stats = 0
        board = utils.initial_board(self.n)
        utils.print_live = self.update_screen

        self.label_status.config(text="cari solusi...", fg="blue")
        start_time = time.time()
        res = utils.run_brute_force(board, 0, self.color_area, self.data_file)
        end_time = time.time()
        self.durasi = int((end_time - start_time) * 1000)

        if res == True:
            for i in range(self.n):
                for j in range(self.n):
                    if board[i][j] == 1:
                        self.block[i][j].config(text="Q", fg="black")
                    else:
                        self.block[i][j].config(text="")

            self.solution_board = board
            self.label_status.config(text="Selesai", fg="green")

            pesan = "Waktu pencarian: " + str(self.durasi) + " ms\n"
            pesan = pesan + "Banyak kasus yang ditinjau: " + str(utils.stats) + "
kasus"
            self.label_hasil.config(text=pesan)

        else:
            self.solution_board = []
            self.label_status.config(text="Tidak ada solusi yang ditemukan.",
fg="red")
            self.label_hasil.config(text="Waktu pencarian: " + str(self.durasi) + "
ms")
            self.label_hasil.config(text=self.label_hasil.cget("text") + "\nBanyak
kasus yang ditinjau: " + str(utils.stats) + " kasus")
```

```
    def save_result(self):
        if self.solution_board == []:
            self.label_status.config(text="error: belum ada solusi untuk disimpan",
fg="red")
            return

        filename = self.input_filename.get()
        if filename == "":
            self.label_status.config(text="error: nama file tidak boleh kosong",
fg="red")
            return

        if filename.endswith(".txt") == False:
            filename = filename + ".txt"

        path = "../test/output/" + filename

        f = open(path, 'w')
        for i in range(self.n):
            str_row = ""
            txt_original = self.data_file[i].strip()
            for j in range(self.n):
                if self.solution_board[i][j] == 1:
                    str_row = str_row + "#"
```

```
                else:
                    if j < len(txt_original):
                        str_row = str_row + txt_original[j]
                    else:
                        str_row = str_row + "."
            f.write(str_row + "\n")

        f.write("\n")
        f.write("Waktu pencarian: " + str(self.durasi) + " ms\n")
        f.write("Banyak kasus yang ditinjau: " + str(utils.stats) + " kasus\n")
        f.close()

        self.label_status.config(text="berhasil disimpan ke " + path, fg="green")
```

```
    def save_image(self):
        if self.solution_board == []:
            self.label_status.config(text="error: belum ada solusi untuk disimpan",
fg="red")
            return

        filename = self.input_filename.get()
        if filename == "":
            self.label_status.config(text="error: nama file tidak boleh kosong",
fg="red")
            return

        if filename.endswith(".txt"):
            filename = filename.replace(".txt", "")

        if filename.endswith(".png"):
            filename = filename.replace(".png", "")

        utils.save_image(self.solution_board, self.data_file, filename, utils.stats,
self.durasi /1000)

        self.label_status.config(text="berhasil disimpan gambar ke
../test/output/image/" + filename + ".png", fg="green")

if __name__ == "__main__":
    app = QueenLinkedIn()
```

Output Image

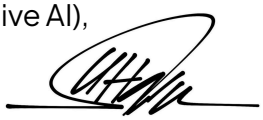
Fungsi save_image ini bertugas memvisualisasikan solusi papan permainan ke dalam file gambar PNG menggunakan library Pillow (PIL), dimulai dengan membuat kanvas putih yang ukurannya dihitung berdasarkan dimensi papan. Setelah itu lalu melakukan iterasi setiap sel matriks untuk menggambar kotak persegi dengan warna background sesuai dengan color palette yang diinput. Jika terdapat block yang berisikan ratu, maka fungsi akan menggambar lingkaran hitam dengan huruf Q.

Sebagai informasi tambahan, terdapat statistik waktu eksekusi dan jumlah kasus yang ditinjau yang ditulis di bagian bawah.

```
def save_image(board, raw_lines, filename, stats, duration):
    n = len(board)
    cell_size = 60
    margin_bot = 100
    width = n * cell_size
    height = n * cell_size + margin_bot

    img = Image.new('RGB', (width, height), color='white')
    draw = ImageDraw.Draw(img)

    color_palette = [
        (255, 179, 186), (255, 223, 186), (255, 255, 186), (186, 255, 201), (186,
225, 255),
        (230, 179, 255), (255, 179, 230), (212, 240, 240), (255, 198, 255), (189,
178, 255),
        (160, 196, 255), (155, 246, 255), (202, 255, 191), (253, 255, 182), (255,
214, 165),
        (255, 173, 173), (200, 200, 200), (240, 230, 140), (176, 224, 230), (144,
238, 144)
    ]

    for r in range(n):
        if r < len(raw_lines):
            str_original = raw_lines[r].strip()
        else:
            str_original = ""*n

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

            if c < len(str_original):
                char = str_original[c]
                color_idx = ord(char)%len(color_palette)
                fill_color = color_palette[color_idx]
            else:
                fill_color = (255, 255, 255)

            draw.rectangle([x0, y0, x1, y1], fill=fill_color, outline='black',
width=2)

            if board[r][c] == 1:
                padding = 10
                draw.ellipse([x0 + padding, y0 + padding, x1 - padding, y1 -
padding], fill='black')

                text_x = x0 + (cell_size // 3)
                text_y = y0 + (cell_size // 4)
                draw.text((text_x, text_y), 'Q', fill='white')
    text_x = 20
    text_y = (n * cell_size) + 20

    text_info = f"Waktu: {int(duration * 1000)} ms\nKasus ditinjau: {stats}"

    draw.text((text_x, text_y), text_info, fill="black")

    img.save(f"../test/output/image/{filename}.png")
```

LAMPIRAN

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



Surya Suharna

Repository Github
https://github.com/suryasuharna23/Tucil1_18223075

Referensi
https://www.w3schools.com/python
https://youtu.be/VgVHFQOeAng?si=eVrk1KYYYprSRnd8
https://pillow.readthedocs.io/en/stable/
https://www.geeksforgeeks.org/python/python-tkinter-tutorial/
https://docs.python.org/3/library/time.html
https://realpython.com/python-use-global-variable-in-function/
https://www.geeksforgeeks.org/python/print-colors-python-terminal/
https://stackoverflow.com/questions/15781802/python-tkinter-clearing-a-frame
https://www.youtube.com/watch?v=StY54VYxdp8&list=PLH42YHDxfBrKnEd3n6JGdWScU01a6FCyI&index=4
and many more...

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	✔	
2	Program berhasil dijalankan	✔	
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	✔	

*Catatan**: Spesifikasi bonus yang tidak diimplementasi adalah 'Program dapat mengisi input dalam bentuk file gambar'.***