LAPORAN TUGAS KECIL 1

PENYELESAIAN PERMAINAN *QUEENS* LINKEDIN

Disusun Oleh :

STEFANI ANGELINE OROH
13524064

MATA KULIAH STRATEGI ALGORITMA (IF2211)
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2026

DAFTAR ISI

DAFTAR GAMBAR

BAB I
PENJELASAN ALGORITMA

Queens LinkedIn adalah permainan logika berbasis papan yang terinspirasi dari permasalahan klasik N-Queens. Permainan ini dimainkan pada papan berukuran $n \times n$, yang dibagi menjadi beberapa wilayah, biasanya ditandai dengan warna yang berbeda-beda. Tujuan permainan adalah menempatkan n queen di papan sehingga tidak ada satupun aturan yang dilanggar. Berikut merupakan beberapa aturan yang harus dipenuhi, antara lain:

1. Pemain harus menempatkan n queen pada papan $n \times n$.
2. Pada setiap baris hanya boleh terdapat tepat 1 (satu) queen.
3. Pada satu kolom hanya boleh terdapat tepat 1 (satu) queen.
4. Dalam satu warna yang sama tidak boleh terdapat lebih dari satu queen.
5. Queen tidak boleh saling bersentuhan secara diagonal, yang berarti queen pada baris yang berurutan tidak boleh berada di kolom yang sama maupun di kolom yang jaraknya hanya satu posisi.

Dalam konteks komputasi, permainan ini dapat diselesaikan dengan menggunakan algoritma yang dapat menghasilkan solusi sesuai dengan aturan yang berlaku.

Algoritma *brute force* atau sering disebut *exhaustive search* merupakan pendekatan penyelesaian masalah dengan cara mencoba seluruh kemungkinan solusi yang ada tanpa pengurangan ruang pencarian (*straightforward*). *Brute force* membutuhkan biaya komputasi yang besar dan waktu yang lama dalam penyelesaiannya dikarenakan ia akan bekerja dengan menghasilkan setiap solusi yang mungkin, kemudian baru diperiksa apakah solusi tersebut memenuhi aturan yang ditentukan dalam suatu permasalahan.

Berbeda dengan pendekatan *brute force* yang membangun seluruh kemungkinan konfigurasi terlebih dahulu sebelum melakukan validasi, pendekatan *backtracking* bekerja lebih efisien dengan melakukan validasi pada setiap tahap pembentukan solusi. Dalam konteks permainan Queens, baik *brute force* maupun *backtracking* sama-sama menempatkan queen satu per satu pada setiap baris secara rekursif. Perbedaanya adalah pada proses pemeriksaan aturan. Pada *backtracking*, setiap kali sebuah queen ditempatkan pada suatu baris, algoritma langsung memeriksa apakah penempatan tersebut melanggar aturan kolom, wilayah, atau kedekatan dengan queen sebelumnya. Jika ditemukan pelanggaran, maka langkah tersebut segera dibatalkan dan algoritma mencoba kolom lain pada baris yang sama

tanpa melanjutkan ke baris berikutnya. Mekanisme penghentian ini disebut dengan *pruning,* yang digunakan untuk memotong cabang pencarian yang sudah dipastikan menghasilkan solusi yang invalid, sehingga tidak membuang waktu pada kombinasi yang sudah pasti gagal. Dengan demikian, *backtracking* yang dilengkapi dengan *pruning* akan mengurangi jumlah ruang pencarian secara signifikan dan menghemat biaya kompleksitas. Meskipun, pada algoritma *brute force* digunakan pendekatan *backtracking* dalam mencari alternatif lain, algoritma ini dapat diklasifikasikan sebagai *pure brute force* karena mencoba semua kombinasi yang mungkin tanpa adanya optimasi yang mengeliminasi langkah-langkah tidak perlu.

Berikut merupakan penjelasan tahapan-tahapan dalam menyelesaikan permasalahan Queens yang dilakukan program, yaitu:

1. Proses dimulai dari pembacaan file .txt yang berisi papan permainan dalam bentuk alphabet A-Z yang mempresentasikan wilayah pada setiap sel papan. Setelah file dibaca, program membentuk sebuah matriks dua dimensi **`grid`** berukuran $n \times n$. Nilai n ditentukan dari jumlah baris pada file tersebut. Jika ukuran board tidak $n \times n$ dan file berisi simbol lain selain alphabet, maka akan keluar pesan kesalahan.
2. Setelah ukuran papan dan grid diperoleh, objek **`QueensSolver`** dibuat dengan menyimpan nilai **`n`**, **`grid`**, dan **`enable_optimization`** (metode pencarian bisa dipilih oleh pengguna antara *brute force* atau *pruning*). Pada tahap ini solver menyiapkan struktur utama, yaitu **`queen_positions`** untuk mencatat posisi queen di setiap baris, **`solution`** untuk menyimpan tampilan papan, serta **`iterations`** untuk menghitung jumlah iterasi. Pada *initial state*, belum ada queen yang diletakkan di papan.
3. Proses penyelesaian diawali dengan pemanggilan fungsi **`solve`**, yang menempatkan queen dari baris pertama. Algoritma bekerja secara rekursif dan bertahap, setiap pemanggilan fungsi akan menempatkan queen pada satu baris tertentu.
4. Pada setiap baris, solver mencoba setiap kemungkinan kolom dari 0 hingga $n - 1$. Untuk setiap percobaan, posisi queen disimpan dalam **`queen_positions`** dan jumlah percobaan dicatat dalam variabel `iterations`. Selama proses pencarian berlangsung, fungsi **`live_update`**

akan menampilkan perkembangan algoritma setiap 10.000 percobaan atau setelah selang waktu 0.3 detik untuk memvisualisasikan bagaimana algoritma bekerja secara bertahap. Total waktu eksekusi dipengaruhi oleh interval pembaruan, jika diperbesar menjadi setiap 50.000 iterasi, maka waktu eksekusi keseluruhan dapat menjadi lebih cepat meskipun algoritma pencariannya tetap sama.

5. Pada program ini, pengguna dapat memilih metode mana yang ingin digunakan, yaitu *brute force* atau *pruning* (optimisasi). Jika mode *brute force* yang digunakan, solver akan terus melanjutkan penempatan queen ke baris berikutnya tanpa melakukan pengecekan apapun sampai semua baris sudah terisi. Setelah lengkap/penuh terbentuk, barulah akan dilakukan validasi secara menyeluruh dengan fungsi **`is_valid_fullboard`**. Sebaliknya, jika mode optimisasi diaktifkan, maka setiap kali queen ditempatkan, akan diperiksa oleh fungsi **`is_valid_sofar`**. Jika terdapat aturan yang dilanggar, solver tidak melanjutkan ke baris berikutnya dan langsung mencoba kolom lain.
6. Pada mode *brute force*, ketika semua baris telah terisi (**`row == n`**) atau kondisi basis terpenuhi, berarti telah terbentuk satu konfigurasi lengkap. Fungsi **`is_valid_fullboard`** akan mengembalikan nilai True jika tidak ada aturan yang dilanggar dan proses berhenti karena solusi telah ditemukan. Namun jika fungsi mengembalikan nilai False, artinya kombinasi tersebut gagal. Oleh karena itu, penempatan queen akan dibatalkan dengan menghapus queen dari posisi yang telah dicoba dan mengatur ulang **`solution`** serta **`queen_positions`**. Setelah itu, solver mencoba kolom berikutnya pada baris yang sama. Proses ini dilakukan berulang kali sampai seluruh kombinasi telah ditelusuri atau ditemukan suatu solusi. Sebagai tambahan, untuk mengantisipasi waktu pencarian yang lama pada metode brute force, program menyediakan fitur STOP untuk menghentikan proses pencarian yang sedang dilakukan.
7. Tampilan hasil akhirnya adalah jumlah iterasi yang dilakukan, lama waktu pencarian (ms), dan visualisasi papan dengan posisi queen yang valid. Setelah solusi ditemukan, pengguna dapat memilih ingin menyimpan hasil dalam bentuk file .txt atau bentuk gambar. Selain itu, pengguna juga dapat kembali

ke halaman awal untuk memasukkan file .txt lain dan melakukan proses pencarian solusi Queens kembali.

Walaupun cara kerja algoritma brute force memastikan solusi untuk setiap masalah, biasanya cara ini dipakai sebagai acuan untuk membandingkan dengan cara lain yang lebih efektif. Ketika menyelesaikan masalah Queens di LinkedIn, penggunaan algoritma brute force akan memerlukan waktu yang cukup lama karena kompleksitasnya akan bertambah secara eksponensial seiring dengan ukuran papan yang dipakai.

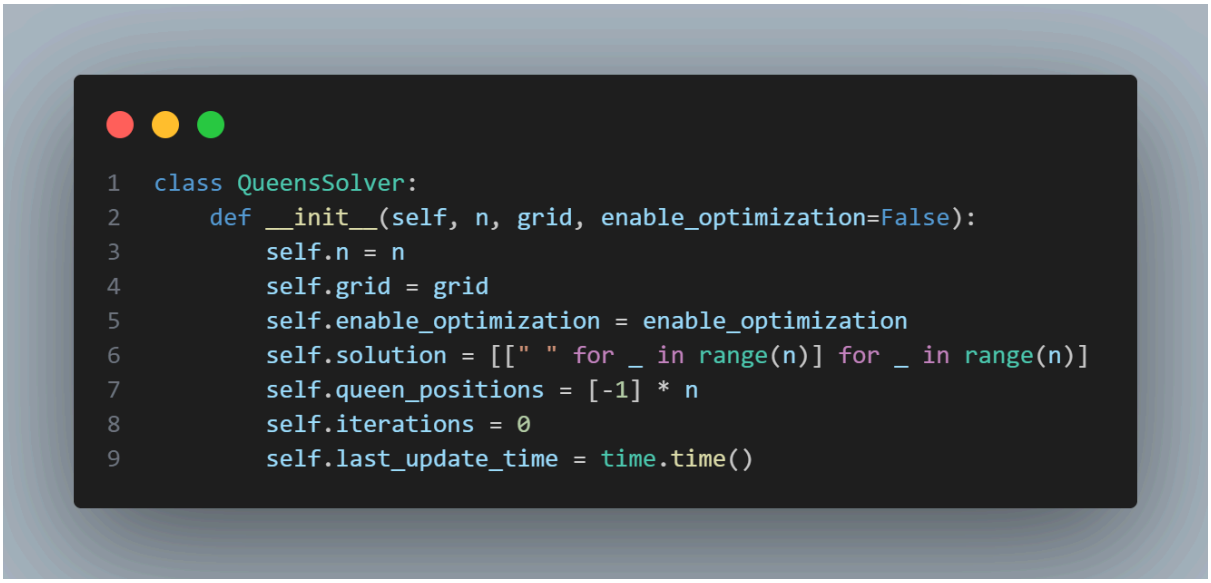
BAB II
SOURCE CODE

Program "Penyelesaian Permainan Queens LinkedIn" diimplementasikan menggunakan bahasa pemrograman Python versi 3.13. Seluruh source code utama, yaitu **`queens_solver.py`** dan **`main_gui.py`** disimpan dalam folder src, sedangkan test case dan hasil disimpan dalam folder test. Aset visual, seperti **`crowns.png`**, **`crowns.ico`**, **`optimization_icon.png`**, dan **`purebf_icon.png`** disimpan dalam folder assets. File utama yang dijalankan adalah **`main_gui.py`**. Program ini tidak menyediakan antarmuka berbasis Command Line Interface (CLI) dikarenakan ketentuan tugas bahwa apabila mahasiswa mengimplementasikan bonus berupa Graphical User Interface (GUI), maka diperbolehkan memilih salah satu bentuk antarmuka saja.

Adapun beberapa library yang digunakan dalam pembuatan program ini, antara lain:

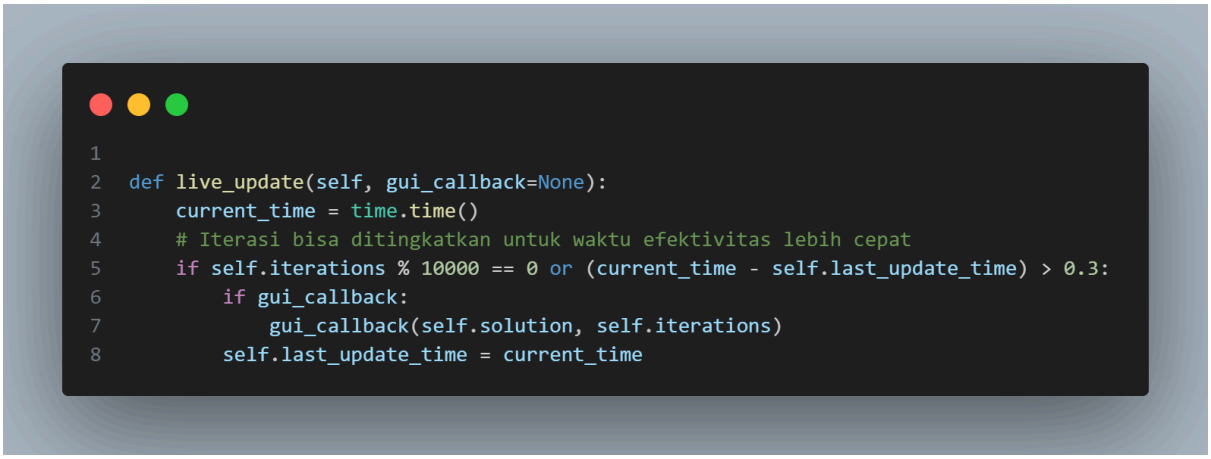
1. tkinter, digunakan untuk membangun elemen dasar GUI, seperti **`Canvas`**, dialog file (**`filedialog`**), dan kotak pesan (**`messagebox`**).
2. customtkinter, merupakan library eksternal berbasis tkinter yang digunakan untuk membuat tampil GUI yang lebih modern.
3. PIL (Pillow), digunakan untuk membaca dan memproses gambar (**`Image`**); mengubah ukuran gambar; menampilkan gambar pada canvas (**`ImageTk`**); dan mengambil screenshot canvas saat hasil disimpan sebagai gambar (**`ImageGrab`**).
4. time, digunakan untuk menghitung waktu pencarian solusi dan mengatur interval pembaruan visualisasi.
5. threading, digunakan agar GUI tetap responsif selama proses pencarian solusi dengan *brute force* atau *backtracking* berlangsung.
6. os, digunakan untuk memanipulasi nama file dan path
7. ctypes, digunakan untuk mengatur DPI awareness pada sistem Windows agar tampilan GUI tidak blur.

Untuk mengimplementasikan algoritma yang telah dijelaskan pada bab sebelumnya, dibuat kelas **`QueenSolver`** dalam file **`queens_solver.py`**. Gambar-gambar berikut menampilkan struktur utama kelas tersebut, termasuk inisialisasi atribut, fungsi validasi, serta fungsi solve yang menjadi inti proses pencarian solusi:



```python
class QueensSolver:
    def __init__(self, n, grid, enable_optimization=False):
        self.n = n
        self.grid = grid
        self.enable_optimization = enable_optimization
        self.solution = [[" " for _ in range(n)] for _ in range(n)]
        self.queen_positions = [-1] * n
        self.iterations = 0
        self.last_update_time = time.time()
```

Gambar 2.1 Struktur Kelas QueensSolver dan Inisialisasi Atribut



```python

def live_update(self, gui_callback=None):
    current_time = time.time()
    # Iterasi bisa ditingkatkan untuk waktu efektivitas lebih cepat
    if self.iterations % 10000 == 0 or (current_time - self.last_update_time) > 0.3:
        if gui_callback:
            gui_callback(self.solution, self.iterations)
        self.last_update_time = current_time
```

Gambar 2.2 Implementasi Fungsi live_update()

Gambar 2.1 menunjukkan struktur awal kelas, termasuk inisialisasi variabel seperti **`n`**, **`grid`**, **`enable_optimization`**, **`solution`**, **`queen_positions`**, **`iterations`**, dan **`last_update_time`**. Variabel-variabel ini digunakan untuk menyimpan ukuran papan, data wilayah, mode pencarian, posisi queen, jumlah iterasi, serta pengaturan waktu pembaruan tampilan. Kemudian, Gambar 2.2 memperlihatkan fungsi **`live_update()`** yang digunakan untuk memperbarui tampilan GUI selama proses pencarian berlangsung. Pembaruan dilakukan setiap 10.000 iterasi atau setelah selang waktu 0,3 detik.

```python

def is_valid_fullboard(self):
    # Periksa kolom
    if len(set(self.queen_positions)) != self.n:
        return False

    # Periksa warna
    used_colors = set()
    for r in range(self.n):
        c = self.queen_positions[r]
        color = self.grid[r][c]
        if color in used_colors:
            return False
        used_colors.add(color)

    # Periksa tetangga diagonal
    for r in range(self.n - 1):
        c1 = self.queen_positions[r]
        c2 = self.queen_positions[r+1]
        if abs(c1 - c2) <= 1:
            return False
    return True

def is_valid_sofar(self, row, col):
    for r in range(row):
        if self.queen_positions[r] == col:
            return False
    current_color = self.grid[row][col]
    for r in range(row):
        if self.grid[r][self.queen_positions[r]] == current_color:
            return False
    if row > 0:
        if abs(self.queen_positions[row-1] - col) <= 1:
            return False
    return True
```

Gambar 2.3 Implementasi Fungsi is_valid_fullboard() dan is_valid_sofar()

Gambar 2.3 menampilkan fungsi validasi yang digunakan untuk memastikan konfigurasi queen memenuhi aturan permainan. Fungsi **`is_valid_fullboard()`** digunakan pada mode brute force, sedangkan **`is_valid_sofar()`** digunakan pada mode optimization

untuk melakukan pruning dengan memeriksa validitas sejak tahap parsial.

```python
def solve(self, row, gui_callback=None, stop_check=None):
    if stop_check and stop_check():
        return False

    # Basis
    if row == self.n:
        if self.enable_optimization:
            return True
        else:
            return self.is_valid_fullboard()

    for col in range(self.n):
        if stop_check and stop_check():
            return False

        self.iterations += 1
        self.live_update(gui_callback)

        self.queen_positions[row] = col
        self.solution[row][col] = "#"

        if self.enable_optimization:
            # Pruning
            if self.is_valid_sofar(row, col):
                if self.solve(row + 1, gui_callback, stop_check):
                    return True
        else:
            # Brute force
            if self.solve(row + 1, gui_callback, stop_check):
                return True

        # Reset (Backtrack)
        self.solution[row][col] = " "
        self.queen_positions[row] = -1

    return False
```

Gambar 2.4 Implementasi Fungsi solve()

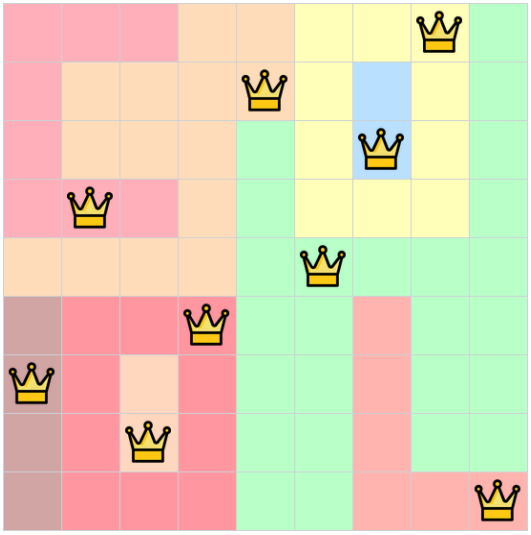
Gambar 2.4 menunjukkan implementasi fungsi **`solve()`** yang merupakan inti dari proses pencarian solusi. Fungsi ini bekerja secara rekursif dengan menempatkan queen pada setiap baris dan mencoba seluruh kemungkinan kolom. Pada bagian basis (**`row == n`**), fungsi menentukan apakah konfigurasi lengkap telah terbentuk. Perbedaan antara mode *brute force*

dan *optimization* terlihat pada percabangan **`enable_optimization`**, di mana mode *optimization* melakukan pengecekan parsial sebelum melanjutkan rekursi.

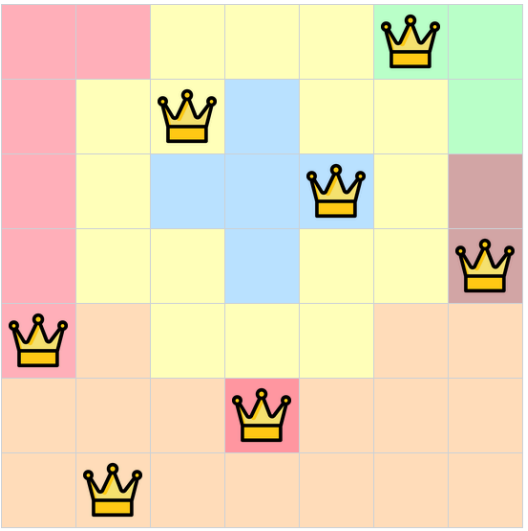
BAB III
HASIL PENGUJIAN

Pada tahap pengujian ini semua test case menggunakan algoritma *brute force*.

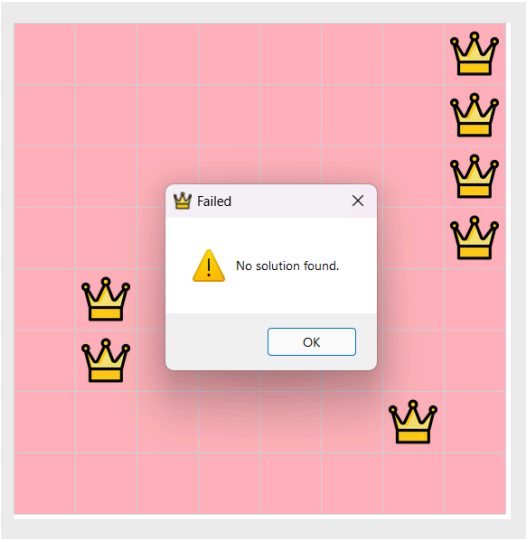
- Test Case 1

Input	Output
AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH	 AAABBCC#D ABBB#CECD ABBBDC#CD A#ABDCCCD BBBBD#DDD FGG#DDHDD #GIGDDHDD FG#GDDHDD FGGGDDHH# Time: 334950 ms Cases: 364209345

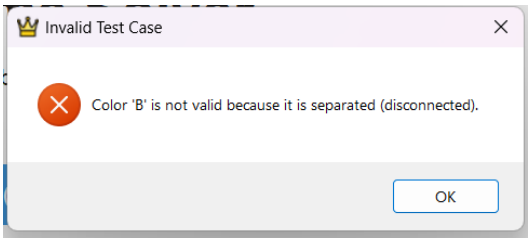
- Test Case 2

Input	Output
AACCCDD ACCECCD ACEEECF ACCECCF ABCCCBB BBBGBBB BBBBBBB	 AACCC#D AC#ECCD ACEE#CF ACCECC# #BCCCBB BBB#BBB B#BBBBB Time: 576 ms Cases: 739137

- Test Case 3

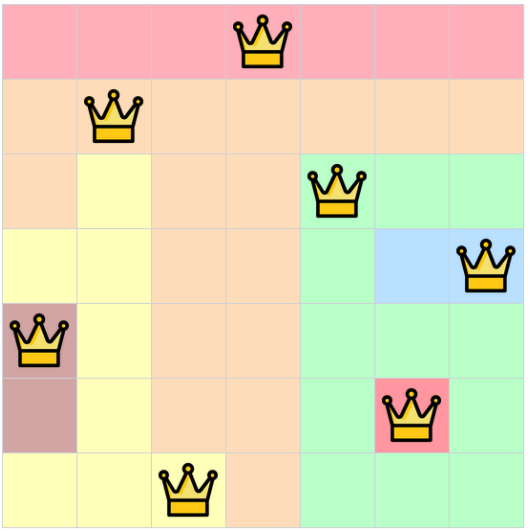
Input	Output
AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA	

- Test Case 4

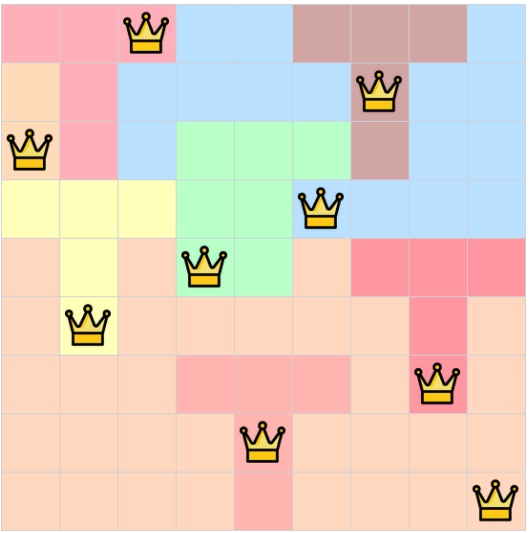
Input	Output
ABCDE BCDEA CDEAB DEABC EABCD	

- Test Case 5

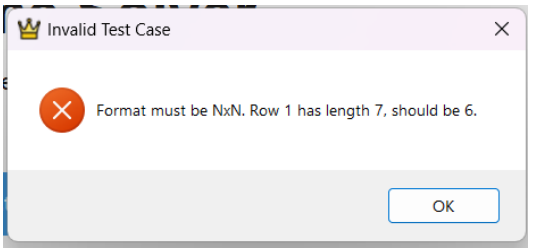
Input	Output

AAAAAAA BBBBBBB BCBBDDD CCBBDEE FCBBDDD FCBBDGD CCCBDDD	 AAA#AAA B#BBBBB BCBB#DD CCBBDE# #CBBDDD FCBBD#D CC#BDDD Time: 471 ms Cases: 445032

- Test Case 6

Input	Output
AAAEEFFFE BAEEEEFEE BAEDDDFEE CCCDDEEEE ICIDDIGGG ICIIIIGI IIIHHHIGI IIIIHIIII IIIIHIIII	 AA#EEFFFE BAEEEE#EE #AEDDDFEE CCCDD#EEE ICI#DIGGG I#IIIIGI IIIHHHI#I IIII#IIII IIIIHIII# Time: 116222 ms Cases: 129495969

- Test Case 7

Input	Output
BBBBBBB BCBBDDD CCBBDEE FCBBDDD FCBBDGD CCCBDDD	

BAB IV
LAMPIRAN

- Link Repository GitHub
  https://github.com/angelineoroh/Tucil1_13524064

- Tabel Pencapaian

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	✓	
2	Program berhasil di jalankan	✓	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	✓	
5	Program memiliki Graphical User Interface (GUI)	✓	
6	Program dapat menyimpan solusi dalam bentuk file gambar	✓	

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

Stefani Angeline Oroh