

Laporan Tugas Kecil 1 IF2211 Strategi Algoritma Semester II Tahun Akademik 2025/2026

Penyelesaian Games LinkedIn “Queens” Menggunakan Algoritma Bruteforce



13524013 - Anindya Naufal Pinasthika

K-01

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025**

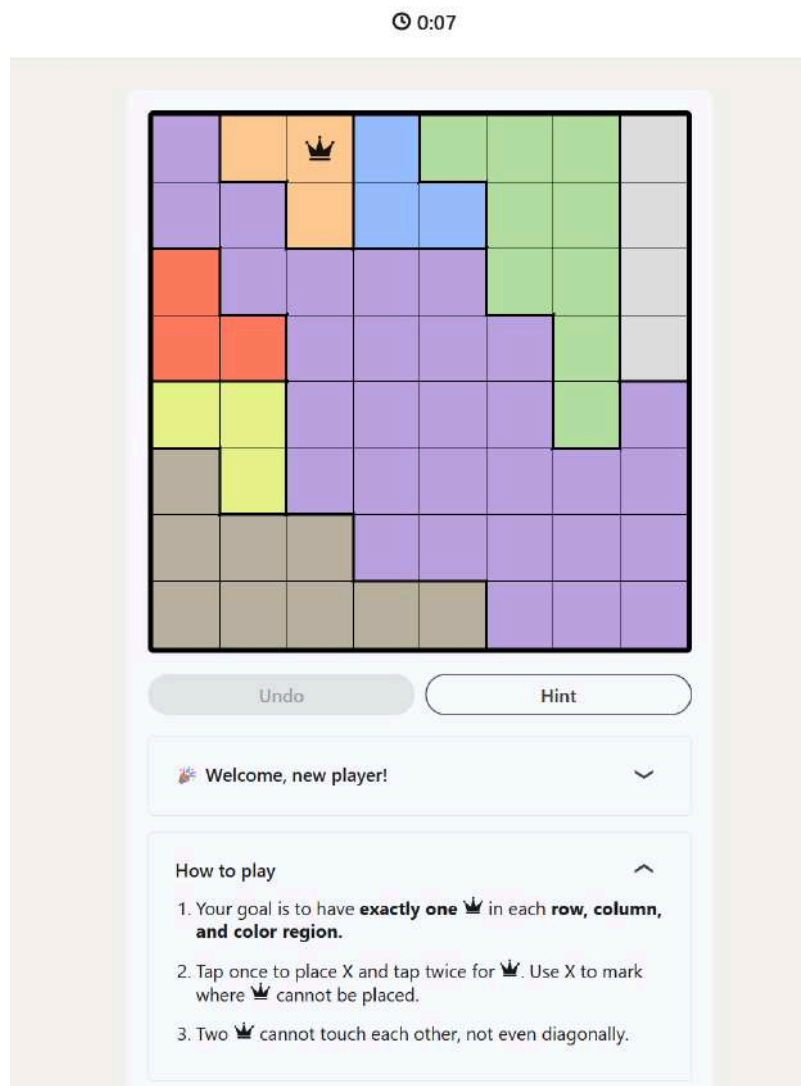
Daftar Isi

Daftar Isi.....	2
1 Pendahuluan.....	3
1.1 Deskripsi Tugas.....	3
Gambar 1 Tampilan permainan game Queens LinkedIn.....	3
Source: https://www.linkedin.com/games/queens/	3
1.2 Tujuan Tugas.....	4
2 Penyelesaian Algoritma dan Alur Program.....	5
2.1 Algoritma Brute Force.....	5
Gambar 2 Visual representasi matriks dalam pendekatan brute force.....	6
2.2 Struktur dan Alur Penyelesaian Program.....	9
Gambar 3 Contoh koordinat matriks queen yang valid pada salah satu game Queens.....	9
3 Struktur Repository dan Source Code.....	10
3.1 Struktur Repository.....	10
3.2 Source Code.....	11
4 Hasil Pengujian.....	34
4.1 Test Case 1 (n = 6 kasus valid).....	34
4.2 Test Case 2 (n = 7 kasus valid).....	35
4.3 Test Case 3 (n = 8 kasus valid).....	36
4.4 Test Case 4 (rows dan column tidak valid).....	37
4.5 Test Case 5 (banyaknya region tidak simetris dengan rows dan column size).....	38
4.6 Test Case 6 (input board valid namun tidak menemukan solusi).....	38
4.7 Test Case 7 (input manual dan mendapatkan solusi yang valid).....	40
5 Lampiran.....	42
- Github Repository.....	42
- Tabel Penilaian.....	42
6 Pernyataan tidak melakukan kecurangan.....	43

1 Pendahuluan

1.1 Deskripsi Tugas

Queens adalah gim logika yang tersedia pada situs jejaring profesional LinkedIn.



Gambar 1 Tampilan permainan game Queens LinkedIn.

Source: <https://www.linkedin.com/games/queens/>

Permainan Queens dimulai dengan sebuah papan kosong dengan warna warna yang kerap disebut “region” dan sebuah Queen. Perlu diketahui bahwa pada game Queens:

Baris = Kolom = Jumlah Region = Jumlah Queen

Sehingga objektifnya hanya dengan menempatkan menetapkan total queen yang sama dengan total region dengan memenuhi aturan berikut:

- Hanya boleh terdapat 1 queen untuk row, column, dan region, jika terdapat lebih dari 1 queen dalam ketentuan tersebut maka placement dianggap tidak valid.
- Queen tidak boleh berdekatan satu sama lain, termasuk untuk arah diagonal. Sehingga jika sebuah queen diletakkan pada satu petak, di sekitar petak tersebut dengan radius 1 kotak tidak valid untuk meletakkan queen selanjutnya

1.2 Tujuan Tugas

Tujuan dari Tugas Kecil 1 IF211 Strategi Algoritma dalam ini adalah dengan menemukan cukup 1 solusi dari permainan Queens LinkedIn dengan menggunakan pendekatan algoritma *brute force*. Di mana dalam definisi brute force sendiri mengharuskan untuk mengecek setiap kasus-kasus yang memungkinkan tanpa ada pendekatan heuristik hingga sebuah kasus telah memenuhi tiap aturan yang ada untuk mencapai solusi tersebut. Jikalau setiap kasus dicek tidak dapat menemukan kasus yang memenuhi semua aturan yang ada, maka games tersebut akan dinilai tidak dapat diselesaikan / tidak dapat menemukan sebuah solusi.

2 Penyelesaian Algoritma dan Alur Program

2.1 Algoritma Brute Force

Pendekatan brute force yang dilakukan memiliki 2 tahap utama, yakni iterasi dan pengecekan. Proses iterasi dilakukan dengan mengecek setiap kombinasi penempatan queen yang mungkin tanpa ada pendekatan heuristik / jalan pintas dalam hal iterasi proses tersebut. Langkah langkah iterasi dilakukan sebagai berikut:

1. Program dimulai dengan mensimulasikan koordinat sebuah queens pada papan size * size (ukuran ini akan bergantung dengan bagaimana user menginputnya) dalam bentuk matriks integer 2D.
2. Pada Matriks yang kosong, pertama-tama semua queen yang ada harus dimasukkan ke dalam papan sebelum melakukan iterasi. Hal ini dimulai dengan memberikan nilai "1" pada matriks sehingga semua queen sudah ditempatkan dalam matriks. Queen awalnya diletakkan pada baris pertama matriks dan akan memanggil rekursif hingga semua queen sudah ada di dalam matriks.
3. Setelah semua queen terinisialisasi. Program akan melakukan pengecekan terhadap matriks tersebut dengan aturan berikut:
 - a. Apakah baris dan kolom hanya terdapat 1 queen yang valid?
 - b. Apakah terdapat queen yang berdekatan satu sama lain dengan radius 1 block (termasuk radius)? Pada kasus ini, kita perlu mengecek semua kemungkinan kasus yang ada seperti bagaimana daerah radius ketika saat berada di pojok, sisi, dan daerah tengah-tengah (daerah tanpa pojok atau sisi)
 - c. Apakah semua region hanya terdapat 1 queen yang valid?

Jika matriks tidak dapat memenuhi aturan tersebut, program akan melakukan proses *backtracking* dengan menggeser queen paling ujung ke petak selanjutnya

4. Ketika queen paling ujung telah melakukan semua pengecekan terhadap tiap kotak di matriks dan melakukan pengecekan serta belum melakukan solusi, queen selanjutnya yang berada sebelum ujung akan menggeser selanjutnya dan queen paling ujung akan berbalik posisi ke tempat yang semula tergeser 1 petak.
5. Proses akan berlanjut hingga program menemukan sebuah solusi yang dapat memenuhi persyaratan semua aturan yang diberikan dan akan memberikan hasilnya. Jika tidak menemukan sebuah solusi, program akan melanjutkan kombinasi penempatan queen hingga kasus terakhir (*worst case scenario*)



Gambar 2 Visual representasi matriks dalam pendekatan brute force

Berikut merupakan psuedocode dari representasi algoritma yang dilakukan:

Brute force iterasi queen

```

procedure bruteForceQueen(input queen: integer, input currIdx: integer, input totalQueen:
integer, input size: integer, input/output temp: array of integer, output success: boolean)

{ bf naive approach ensure all possible combination of the queen is tested }

{ base case }
if queen = totalQueen then
    casesEvaluated ← casesEvaluated + 1
    checkQueen(temp, matrix, size, output isValid)
    if isValid then
        success ← true
        → { keluar dari prosedur }
    success ← false
    → { keluar dari prosedur }

totalSize ← size * size

i traversal [currIdx..totalSize-1]
    row ← i div size
    col ← i mod size

    temp[row][col] ← 1

    bruteForceQueen(queen + 1, i + 1, totalQueen, size, temp, output result)
    if result then
        success ← true
        → { keluar dari prosedur }

    temp[row][col] ← 0

success ← false
→ { keluar dari prosedur }

```

Pengecekan validasi penempatan queen

procedure checkQueen(input temp: array of integer, input matrix: array of char, input size: integer, output isValid: boolean)

```
{ edge case: n = 1 is always right for [0,0] }
{ another thing to note: n == 2 && n == 3 is mathematically impossible }
if size = 1 then
    isValid ← true
    → { keluar dari prosedur }

counter ← 0

{ check horizontal }
i traversal [0..size-1]
    j traversal [0..size-1]
        if temp[i][j] = 1 then
            counter ← counter + 1
    if counter > 1 then
        isValid ← false
        → { keluar dari prosedur }
    counter ← 0

{ check vertical }
i traversal [0..size-1]
    j traversal [0..size-1]
        if temp[j][i] = 1 then
            counter ← counter + 1
    if counter > 1 then
        isValid ← false
        → { keluar dari prosedur }
    counter ← 0

{ check surrounding }
i traversal [0..size-1]
    j traversal [0..size-1]
        if temp[i][j] = 1 then
            { case: corner }
            if i = 0 and j = 0 then
                if temp[i][j+1] = 1 or temp[i+1][j+1] = 1 or temp[i+1][j] = 1 then
                    isValid ← false
                    → { keluar dari prosedur }
            else if i = 0 and j = size - 1 then
                if temp[i][j-1] = 1 or temp[i+1][j-1] = 1 or temp[i+1][j] = 1 then
                    isValid ← false
                    → { keluar dari prosedur }
            else if i = size - 1 and j = size - 1 then
                if temp[i][j-1] = 1 or temp[i-1][j-1] = 1 or temp[i-1][j] = 1 then
                    isValid ← false
                    → { keluar dari prosedur }
            else if i = size - 1 and j = 0 then
                if temp[i-1][j] = 1 or temp[i-1][j+1] = 1 or temp[i][j+1] = 1 then
                    isValid ← false
                    → { keluar dari prosedur }
```

```

    { case: edge (without corners) }
    if i = 0 and j > 0 and j < size-1 then
        if temp[i][j+1] = 1 or temp[i+1][j+1] = 1 or temp[i+1][j] = 1 or temp[i+1][j-1] = 1 or
temp[i][j-1] = 1 then
            isValid ← false
            → { keluar dari prosedur }
        else if i = size - 1 and j > 0 and j < size-1 then
            if temp[i][j-1] = 1 or temp[i-1][j-1] = 1 or temp[i-1][j] = 1 or temp[i-1][j+1] = 1 or
temp[i][j+1] = 1 then
                isValid ← false
                → { keluar dari prosedur }
            else if j = size - 1 and i > 0 and i < size-1 then
                if temp[i-1][j] = 1 or temp[i-1][j-1] = 1 or temp[i][j-1] = 1 or temp[i+1][j-1] = 1 or
temp[i+1][j] = 1 then
                    isValid ← false
                    → { keluar dari prosedur }
                else if j = 0 and i > 0 and i < size-1 then
                    if temp[i-1][j] = 1 or temp[i-1][j+1] = 1 or temp[i][j+1] = 1 or temp[i+1][j+1] = 1 or
temp[i+1][j] = 1 then
                        isValid ← false
                        → { keluar dari prosedur }

    { case: any place (not in edge or corner) }
    if i > 0 and i < size - 1 and j > 0 and j < size - 1 then
        if temp[i][j-1] = 1 or temp[i+1][j-1] = 1 or temp[i+1][j] = 1 or temp[i+1][j+1] = 1 or
temp[i][j+1] = 1 or temp[i-1][j+1] = 1 or temp[i-1][j] = 1 or temp[i-1][j-1] = 1 then
            isValid ← false
            → { keluar dari prosedur }

{ check block }
arrCheck array of char size size
arrCount ← 0

i traversal [0..size-1]
    j traversal [0..size-1]
        if temp[i][j] = 1 then
            arrCheck[arrCount] ← matrix[i][j]
            arrCount ← arrCount + 1

i traversal [0..arrCount-1]
    j traversal [i+1..arrCount-1]
        if arrCheck[i] = arrCheck[j] then
            isValid ← false
            → { keluar dari prosedur }

isValid ← true
→ { keluar dari prosedur }

```

2.2 Struktur dan Alur Penyelesaian Program

Program akan memulai dengan meminta sebuah input. Input dapat dilakukan dengan 2 cara, yakni dengan input menggunakan .txt, dan input manual dalam program GUI, di mana user dapat menggunakan keyboard untuk membuat size dan isi board yang diinginkan.

Program selanjutnya akan melewati beberapa validasi input, dengan mengecek:

- Apakah dimensi rows dan column simetris?
- Apakah jumlah daerah simetris terhadap column dan rows?

Jika input yang diberikan sudah valid untuk aturan tersebut, input akan diubah yang awalnya String input menjadi sebuah matrix 2D of character. Matrix 2D character akan memunculkan size matriksnya di mana akan membuat sebuah matriks 2D of integer yang akan menjadi representasi penempatan queen dalam bentuk koordinat matriks 2D. Matriks koordinat ini yang akan diproses dalam algoritma brute force sehingga mendapatkan koordinat penempatan yang valid.

$$\begin{bmatrix} 1 & - & - & - & - \\ - & - & 1 & - & - \\ - & - & - & - & 1 \\ - & 1 & - & - & - \\ - & - & - & -1 & - \end{bmatrix}$$

Gambar 3 Contoh koordinat matriks queen yang valid pada salah satu game Queens

Matriks koordinat yang sudah valid akan ditransformasikan dalam Matrix 2D character untuk menempatkan daerah dengan koordinat yang memiliki nilai “1” pada matriks koordinat, untuk diganti dengan “#” (jika dalam GUI, akan membuat sebuah canvas di mana pada grid dengan koordinat matriks queen yang valid, akan ditempatkan sebuah gambar queen).

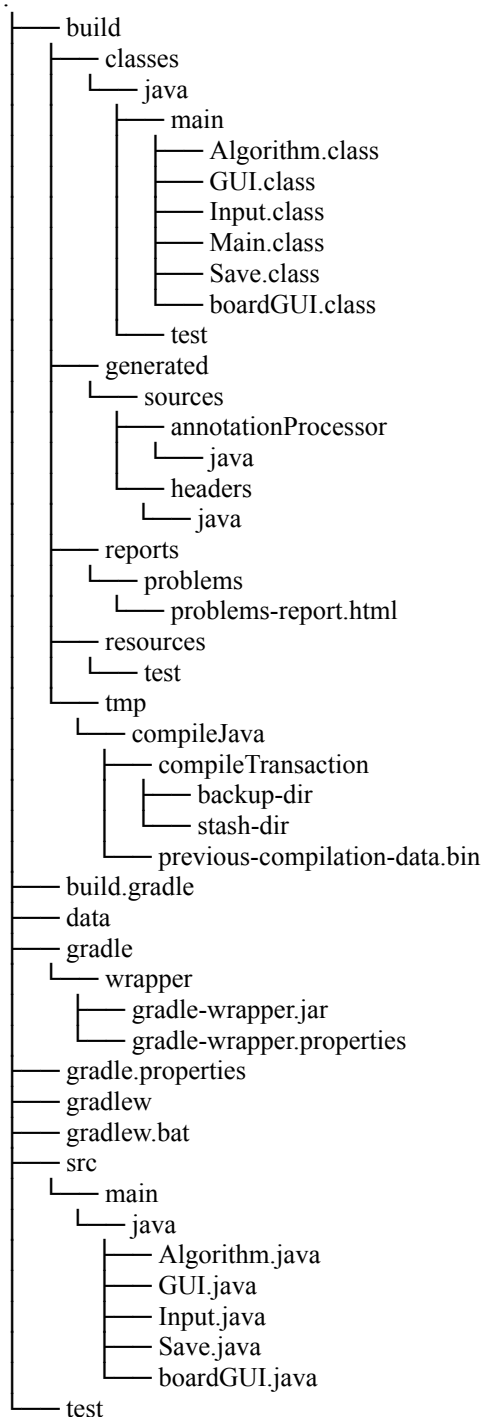
Pada proses melakukan algoritma, program akan melakukan multithreading di mana terdapat thread yang melakukan proses brute force, serta terdapat thread yang akan menampilkan proses brute force, di mana setiap 0.5 detik, kondisi matriks yang sedang di brute force akan ditampilkan di GUI, menampilkan proses brute force yang sedang terjadi.

Setelah eksekusi brute force dilakukan, akan menampilkan solusi (jika menemukan sebuah solusi), dan akan memberikan opsi pilihan untuk menyimpan gambar dalam bentuk .txt atau image.

3 Struktur Repository dan Source Code

3.1 Struktur Repository

Berikut merupakan struktur repository dari source code. Input file dalam bentuk .txt bisa dimasukkan di mana saja, namun disarankan di ./data, sedangkan hasil dari penyelesaian algoritma disimpan di ./test.



Struktur repository memiliki beberapa folder yang disimpan:

- build = folder binary yang dari Java Gradle
- data = folder untuk menyimpan .txt
- gradle = wrapper gradle untuk keperluan GUI
- src = source code program dalam bahasa pemrograman Java

- test = folder yang menjadi output dari hasil penyelesaian program

3.2 *Source Code*

Berikut source code yang terlampir dalam repository:

Algorithm.java

```
public class Algorithm {
    public int queens;
    public char[][] matrix;
    public int[][] temp;
    public long casesEvaluated = 0;

    public volatile boolean isRunning = false;
    public volatile boolean isFound = false;

    public Algorithm (char[][] matrix, int size) {
        this.queens = size;
        this.matrix = matrix;
        this.temp = new int[size][size];
    }

    // brute force by placing every queen horizontally
    //and increment to move down 1 queen at a time

    public void initializeTemp(int size) {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                this.temp[i][j] = 0;
            }
        }
    }

    public void printProcessQueen(int size, char[][] matrix,
int[][] temp) {
        // setQueen(temp, matrix, size);

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (temp[i][j] == 1) {
                    System.out.print("# ");
                } else {
                    System.out.print(matrix[i][j] + " ");
                }
            }
        }
    }
}
```

```

        System.out.println();
    }
    System.out.println();
}
public void printFinalQueen(int size, char[][] matrix, int[][]
temp) {
    setQueen(temp, matrix, size);
    System.out.println("Final Result: \n");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            System.out.print(matrix[i][j] + " ");
        }
        System.out.println();
    }
}

public boolean processBruteForceQueen(int size, int[][] temp,
char[][] matrix) {
    initializeTemp(size);
    this.casesEvaluated = 0;

    this.isRunning = true;
    this.isFound = false;

    boolean flagResult = false;

    flagResult = bruteForceQueen(0, 0, size, size, temp);

    this.isRunning = false;
    this.isFound = flagResult;

    if (flagResult) {
        printFinalQueen(size, matrix, temp);
        return true;
    } else {
        return false;
    }
}

public boolean bruteForceQueen(int queen, int currIdx, int
totalQueen, int size, int[][] temp) {
    //bf naive approach O(c(n^2, n)) ensure all possible

```

combination of the queen is tested

```
        if (queen == totalQueen) {
            this.casesEvaluated++;
            if (checkQueen(this.temp, this.matrix, size)) {
                return true;
            }
            return false;
        }

        int totalSize = size*size;

        for (int i = currIdx; i < totalSize; i++) {
            int row = i / size;
            int col = i % size;

            this.temp[row][col] = 1;
            if (bruteForceQueen(queen+1, i+1, totalQueen, size,
temp)) {
                return true;
            }

            this.temp[row][col] = 0;
        }

        return false;
    }

    public boolean checkQueen(int[][] temp, char[][] matrix, int
size) {

        // edge case: n = 1 is always right for [0,0]
        // another thing to note: n == 2 && n == 3 is
mathematically impossible
        if (size == 1) {
            return true;
        }

        int counter = 0;

        // check horizontal
```

```

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++){
                if (temp[i][j] == 1) {
                    counter++;
                }
            }
            if (counter > 1) {
                return false;
            }
            counter = 0;
        }

        // check vertical
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++){
                if (temp[j][i] == 1) {
                    counter++;
                }
            }
            if (counter > 1) {
                return false;
            }
            counter = 0;
        }

        // check surrounding
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (temp[i][j] == 1) {
                    // case: corner
                    if (i == 0 && j == 0) {
                        if (temp[i][j+1] == 1 || temp[i+1][j+1] ==
1 || temp[i+1][j] == 1) {
                            return false;
                        }
                    } else if (i == 0 && j == size - 1) {
                        if (temp[i][j-1] == 1 || temp[i+1][j-1] ==
1 || temp[i+1][j] == 1) {
                            return false;
                        }
                    } else if (i == size - 1 && j == size - 1) {
                        if (temp[i][j-1] == 1 || temp[i-1][j-1] ==
1 || temp[i-1][j] == 1) {
                            return false;
                        }
                    }
                }
            }
        }

```

```

    }
    } else if (i == size - 1 && j == 0) {
        if (temp[i-1][j] == 1 || temp[i-1][j+1] ==
1 || temp[i][j+1] == 1) {
            return false;
        }
    }

    // case: edge (without corners)
    if (i == 0 && j > 0 && j < size-1) {
        if (temp[i][j+1] == 1 || temp[i+1][j+1] ==
1 || temp[i+1][j] == 1 || temp[i+1][j-1] == 1 || temp[i][j-1] == 1) {
            return false;
        }
    } else if (i == size - 1 && j > 0 && j <
size-1) {
        if (temp[i][j-1] == 1 || temp[i-1][j-1] ==
1 || temp[i-1][j] == 1 || temp[i-1][j+1] == 1 || temp[i][j+1] == 1) {
            return false;
        }
    } else if (j == size - 1 && i > 0 && i <
size-1) {
        if (temp[i-1][j] == 1 || temp[i-1][j-1] ==
1 || temp[i][j-1] == 1 || temp[i+1][j-1] == 1 || temp[i+1][j] == 1) {
            return false;
        }
    } else if (j == 0 && i > 0 && i < size-1) {
        if (temp[i-1][j] == 1 || temp[i-1][j+1] ==
1 || temp[i][j+1] == 1 || temp[i+1][j+1] == 1 || temp[i+1][j] == 1) {
            return false;
        }
    }

    // case: any place (not in edge or corner)
    if (i > 0 && i < size - 1 && j > 0 && j < size
- 1) {
        if (temp[i][j-1] == 1 || temp[i+1][j-1] ==
1 || temp[i+1][j] == 1 || temp[i+1][j+1] == 1
|| temp[i][j+1] == 1 || temp
[i-1][j+1] == 1 || temp[i-1][j] == 1 || temp[i-1][j-1] == 1) {

            return false;
        }
    }

```

```

    }

    }

    }

    // check block
    char[] arrCheck = new char[size];
    int arrCount = 0;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (temp[i][j] == 1) {
                arrCheck[arrCount] = matrix[i][j];
                arrCount++;
            }
        }
    }

    for (int i = 0; i < arrCheck.length; i++) {
        for (int j = i + 1; j < arrCheck.length; j++) {
            if (arrCheck[i] == arrCheck[j]) {
                return false;
            }
        }
    }

    return true;
}

public void setQueen(int[][] temp, char[][] matrix, int size)
{
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (temp[i][j] == 1) {
                matrix[i][j] = '#';
            }
        }
    }
}

```



```

        public boolean bruteForce (char[][] matrix, int size, int
queens) {
            // initialize queen to be put
            boolean flag = processBruteForceQueen(size, temp, matrix);

            return flag;
        }
    }
}

```

boardGUI.java

```

import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.TextAlignment;

public class boardGUI {

    private final Color[] colors = {
        Color.web("#E74C3C"),
        Color.web("#8E44AD"),
        Color.web("#3498DB"),
        Color.web("#16A085"),
        Color.web("#F39C12"),
        Color.web("#D35400"),
        Color.web("#C0392B"),
        Color.web("#27AE60"),
        Color.web("#2980B9"),
        Color.web("#2C3E50"),
        Color.web("#E67E22"),
        Color.web("#1ABC9C"),
        Color.web("#9B59B6"),
        Color.web("#2ECC71"),
        Color.web("#34495E"),
        Color.web("#7F8C8D"),
        Color.web("#C2185B"),
        Color.web("#512E5F"),
        Color.web("#154360"),
        Color.web("#0E6251"),
        Color.web("#784212"),
    }
}

```

```

        Color.web("#7B241C"),
        Color.web("#1B4F72"),
        Color.web("#641E16"),
        Color.web("#4A235A"),
        Color.web("#0B5345")
    };

    public int charToColor (char[][] matrix, int i, int j) {

        char region = matrix[i][j];
        int colorIndex = Character.toUpperCase(region) - 'A';
        colorIndex = colorIndex % 26;
        return colorIndex;

    }

    public Canvas board (char[][] matrix, int[][] temp, int size) {
        int cellSize = 100;
        int cellWidth = size * cellSize;
        int cellHeight = size * cellSize;

        Canvas canvas = new Canvas(cellWidth, cellHeight);
        GraphicsContext gc = canvas.getGraphicsContext2D();

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                double x = j * cellSize;
                double y = i * cellSize;
                int colorIndex = charToColor(matrix, i, j);

                // color region
                gc.setFill(colors[colorIndex]);
                gc.fillRect(x, y, cellSize, cellSize);

                //border
                gc.setStroke(Color.BLACK);
                gc.setLineWidth(2);
                gc.strokeRect(x, y, cellSize, cellSize);

                // queen
                if (temp[i][j] == 1) {

```

```

        gc.setFill(Color.GOLD);
        gc.setFont(Font.font("Segoe UI Emoji",
FontWeight.BOLD, cellSize * 0.6));
        gc.setTextAlign(TextAlignment.CENTER);

        gc.fillText("👑", x + (cellSize / 2.0), y +
(cellSize / 1.4));

        gc.setEffect(null);
    }

}

return canvas;
}
}

```

GUI.java

```

import java.io.File;
import javafx.util.Duration;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Spinner;
import javafx.scene.control.TextField;
import javafx.scene.control.TextFormatter;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

```

```

public class GUI extends Application {

    private Stage primaryStage;
    private VBox resultContainer;
    private Label sizeLabel;
    private FileChooser scanTxtFile;
    private boardGUI visualizerHelper = new boardGUI();

    @Override
    public void start(Stage stage) {
        this.primaryStage = stage;

        Label mainLabel = new Label("Queens LinkedIn Solver");
        mainLabel.setStyle("-fx-font-size: 30px; -fx-font-weight:
bold;");

        sizeLabel = new Label("Upload the queens board in .txt format
to begin\n                        Choose input method:");
        sizeLabel.setStyle("-fx-font-size: 24px;");

        scanTxtFile = new FileChooser();
        scanTxtFile.setTitle("Open .txt file");
        scanTxtFile.setInitialDirectory(new
File(System.getProperty("user.home")));
        scanTxtFile.getExtensionFilters().addAll(
            new FileChooser.ExtensionFilter("Text Files", "*.txt")
        );

        resultContainer = new VBox();
        resultContainer.setAlignment(Pos.CENTER);
        resultContainer.setPadding(new Insets(20));

        // upload .txt as input
        Button uploadButton = new Button("Upload File (.txt)");
        uploadButton.setPrefSize(150, 40);

        //manual input
        Button manualButton = new Button("Manual Input");
        manualButton.setPrefSize(150, 40);

        HBox buttonBox = new HBox(20);
        buttonBox.setAlignment(Pos.CENTER);
        buttonBox.getChildren().addAll(uploadButton, manualButton);
    }
}

```

```

        // file .txt input
        uploadButton.setOnAction(e -> handleFileUpload());

        // manual input
        manualButton.setOnAction(e -> showSizeSelector());

        VBox topContainer = new VBox(20);
        topContainer.setAlignment(Pos.CENTER);
        topContainer.setPadding(new Insets(50, 0, 0, 0));
        topContainer.getChildren().addAll(mainLabel, sizeLabel,
buttonBox);

        BorderPane root = new BorderPane();
        root.setTop(topContainer);
        root.setCenter(resultContainer);

        Scene scene = new Scene(root, 1000, 800);
        stage.setScene(scene);

        stage.maximizedProperty().addListener((observable, oldValue,
newValue) -> {
            if (newValue) {
                stage.setFullScreen(true);
            }
        });

        stage.show();
    }

    private void handleFileUpload() {
        boolean isFullScreen = primaryStage.isFullScreen();
        File selectedFile = scanTxtFile.showOpenDialog(primaryStage);

        if (isFullScreen) {
            Platform.runLater(() -> primaryStage.setFullScreen(true));
        }

        if (selectedFile != null) {
            resultContainer.getChildren().clear();

            Input fileInput = new Input();
            fileInput.processFileFromGUI(selectedFile);

```

```

        String validation = fileInput.inputValidation();
        if (validation != null){
            Label errorLabel = new Label(validation);
            errorLabel.setStyle("-fx-text-fill: red;
-fx-font-size: 18px; -fx-font-weight: bold;");
            resultContainer.getChildren().add(errorLabel);
            return;
        }

        char[][] matrix =
fileInput.inputToMatrix(fileInput.savedInputString);
        startSolver(matrix, fileInput.size,
selectedFile.getName());
    }
}

private void showSizeSelector() {
    resultContainer.getChildren().clear();
    sizeLabel.setText("Select Board Size:");

    // set max to 26 yo match the total of alphabet
    Spinner<Integer> sizeSpinner = new Spinner<>(1, 26, 5);
    sizeSpinner.setStyle("-fx-font-size: 18px;");
    sizeSpinner.setPrefWidth(100);

    Button nextButton = new Button("Create Board");

    nextButton.setOnAction(e -> {
        int size = sizeSpinner.getValue();
        showManualGrid(size);
    });

    VBox selectorBox = new VBox(20, sizeSpinner, nextButton);
    selectorBox.setAlignment(Pos.CENTER);
    resultContainer.getChildren().add(selectorBox);
}

private void showManualGrid(int size) {
    resultContainer.getChildren().clear();
    sizeLabel.setText("Enter Regions (A-Z):");

    GridPane grid = new GridPane();

```

```

grid.setAlignment(Pos.CENTER);
grid.setHgap(5);
grid.setVgap(5);

TextField[][] inputs = new TextField[size][size];

// create size * size grid
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        TextField tf = new TextField();
        tf.setPrefSize(40, 40);
        tf.setAlignment(Pos.CENTER);
        tf.setStyle("-fx-font-weight: bold; -fx-border-color:
gray;");

        // force the input to capital letter & 1 char so that
its consistent with .txt input
        tf.setTextFormatter(new TextFormatter<>(change -> {
            String newText = change.getControlNewText();
            if (newText.length() > 1){
                return null;
            }
            return change;
        }));
        tf.textProperty().addListener((obs, oldVal, newVal) ->
{
            if (!newVal.isEmpty()) {
                String upper = newVal.toUpperCase();
                tf.setText(upper);
            } else {
                tf.setStyle("-fx-control-inner-background:
white; -fx-border-color: gray;");
            }
        });

        inputs[i][j] = tf;
        grid.add(tf, j, i);
    }
}

Button solveButton = new Button("Solve Puzzle");
solveButton.setPadding(new Insets(10, 30, 10, 30));

```

```

Label errorMsg = new Label("");
errorMsg.setStyle("-fx-text-fill: red;");

solveButton.setOnAction(e -> {
    StringBuilder sb = new StringBuilder();
    try {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                String txt = inputs[i][j].getText();
                if (txt == null || txt.isEmpty() ||
!txt.matches("[A-Z]")) {
                    throw new Exception("Please fill all cells
with A-Z");
                }
                sb.append(txt);
            }
            if (i < size - 1) sb.append("\n");
        }

        Input manualInput = new Input();
        manualInput.size = size;
        manualInput.savedInputString = sb.toString();

        String valid = manualInput.inputValidation();
        if (valid != null) {
            errorMsg.setText(valid);
            return;
        }

        char[][] matrix =
manualInput.inputToMatrix(manualInput.savedInputString);
        startSolver(matrix, size, "ManualInput.txt");

    } catch (Exception ex) {
        errorMsg.setText("Error: " + ex.getMessage());
    }
});

VBox container = new VBox(20, grid, errorMsg, solveButton);
container.setAlignment(Pos.CENTER);
resultContainer.getChildren().add(container);
}

```



```

private void startSolver(char[][] matrix, int size, String
fileName) {
    resultContainer.getChildren().clear();
    sizeLabel.setText("");

    // Prepare Brute Force Matrix
    char[][] bruteForceMatrix = new char[size][size];
    for (int i = 0; i < size; i++) {
        System.arraycopy(matrix[i], 0, bruteForceMatrix[i], 0,
size);
    }

    Label resultLabel = new Label("Result: ");
    resultLabel.setStyle("-fx-font-size: 24px;");
    Label statusLabel = new Label("Searching...");
    Label totalExecutionLabel = new Label("");

    Canvas initialCanvas = visualizerHelper.board(matrix, new
int[size][size], size);

    VBox.setMargin(resultLabel, new Insets(0, 0, 20, 0));

    VBox.setMargin(statusLabel, new Insets(20, 0, 5, 0));

    VBox.setMargin(totalExecutionLabel, new Insets(0, 0, 20, 0));

    resultContainer.getChildren().addAll(resultLabel,
initialCanvas, statusLabel, totalExecutionLabel);

    Algorithm queenBoard = new Algorithm(bruteForceMatrix, size);

    Timeline timelineVisualizer = new Timeline(new
KeyFrame(Duration.seconds(0.5), event -> {
        if (queenBoard.isRunning) {
            if(resultContainer.getChildren().size() > 1) {
                resultContainer.getChildren().remove(1);
                Canvas snapshot = visualizerHelper.board(matrix,
queenBoard.temp, size);
                resultContainer.getChildren().add(1, snapshot);
            }
        }
    }));
    timelineVisualizer.setCycleCount(Timeline.INDEFINITE);

```

```

        timelineVisualizer.play();

        Thread workerThread = new Thread(() -> {
            long startTime = System.currentTimeMillis();
            boolean success = queenBoard.bruteForce(bruteForceMatrix,
size, size);
            long endTime = System.currentTimeMillis();
            long executionTime = endTime - startTime;

            Platform.runLater(() -> {
                timelineVisualizer.stop();

                resultContainer.getChildren().remove(1);
                Canvas finalCanvas = visualizerHelper.board(matrix,
queenBoard.temp, size);
                resultContainer.getChildren().add(1, finalCanvas);

                if (success) {
                    resultLabel.setText("Result: Solved");
                    statusLabel.setText("Waktu pencarian: " +
executionTime + " ms");
                    totalExecutionLabel.setText("Banyak kasus yang
ditinjau: " + queenBoard.casesEvaluated + " kasus");
                    addSaveButtons(matrix, queenBoard.temp,
bruteForceMatrix, size, fileName);
                } else {
                    resultLabel.setText("Result: No Solution Found");
                    statusLabel.setText("Waktu pencarian: " +
executionTime + " ms");
                    totalExecutionLabel.setText("Banyak kasus yang
ditinjau: " + queenBoard.casesEvaluated + " kasus");
                }
            });
        });

        workerThread.setDaemon(true);
        workerThread.start();
    }

    private void addSaveButtons(char[][] matrix, int[][] temp,
char[][] resultMatrix, int size, String fileName) {
        HBox saveButtonContainer = new HBox(20);
        saveButtonContainer.setAlignment(Pos.CENTER);

```

```

VBox.setMargin(saveButtonContainer, new Insets(10, 0, 0, 0));
Button saveAsTxt = new Button("Save as .TXT");
saveAsTxt.setOnAction(ev -> {
    Save txtSaver = new Save();
    txtSaver.saveResult(resultMatrix, size, fileName);
    saveAsTxt.setText("Saved");
    saveAsTxt.setDisable(true);
});

Button saveAsImage = new Button("Save as Image");
saveAsImage.setOnAction(ev -> {
    Save imgSaver = new Save();
    imgSaver.saveToImage(matrix, temp, size, fileName);
    saveAsImage.setText("Saved");
    saveAsImage.setDisable(true);
});

saveButtonContainer.getChildren().addAll(saveAsTxt,
saveAsImage);
resultContainer.getChildren().add(saveButtonContainer);
}

public static void main(String[] args) {
    launch();
}
}

```

Input.java

```

import java.io.File;
import java.util.Set;
import java.util.HashSet;
import java.util.Scanner;
import java.io.FileNotFoundException;

public class Input {
    public String savedInputString = "";

    public char[][] inputMatrix;
    public int size;

    public void inputStringConstructor(String input) {
        savedInputString += input;
    }
}

```

```
        savedInputString += "\n";
    }

    public void processInput(String fileName) {
        try {
            File inputObj = new File("data/" + fileName);

            Scanner read = new Scanner(inputObj);

            while (read.hasNextLine()) {
                String input = read.nextLine();

                // since the input is always n x n, take column value
                // to create a matrix n x n afterward;

                this.size = input.length();

                inputStringConstructor(input);

            }

            read.close();

        } catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    public void processFileFromGUI(File file) {
        try{
            Scanner read = new Scanner(file);

            this.savedInputString = "";

            if (read.hasNextLine()) {
                String firstLine = read.nextLine();

                this.size = firstLine.length();
                inputStringConstructor(firstLine);
            }
        }
```

```

        while (read.hasNextLine()) {
            String input = read.nextLine();
            inputStringConstructor(input);
        }
        read.close();

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

public char[][] crateMatrix () {
    this.inputMatrix = new char[size][size];
    return this.inputMatrix;
}

public char[][] inputToMatrix (String input) {
    String[] temp = savedInputString.split("\n");

    this.inputMatrix = crateMatrix();
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++){
            inputMatrix[i][j] = temp[i].charAt(j);
        }
    }

    return inputMatrix;
}

public String inputValidation () {
    //check row & col
    String temp[] = savedInputString.split("\n");
    int row = temp.length;
    int col = temp[0].trim().length();

    if (row != col){
        return "Panjang dan lebar tidak simetris";
    }

    for (int i = 0; i < row; i++) {
        if (temp[i].trim().length() != col) {
            return "Panjang dan lebar tidak simetris";
        }
    }
}

```

```

    }

    //check region

    char[][] tempMatrix = inputToMatrix(savedInputString);

    Set<Character> regions = new HashSet<>();

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            regions.add(tempMatrix[i][j]);
        }
    }

    if (regions.size() != row || regions.size() != col){
        return "Daerah tidak simetris dengan dimensi papan";
    }

    return null;
}
}

```

Save.java

```

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import javax.imageio.ImageIO;

import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.WritableImage;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.TextAlignment;
import javafx.embed.swing.SwingFXUtils;

public class Save {

```

```
private final Color[] colors = {
    Color.web("#E74C3C"),
    Color.web("#8E44AD"),
    Color.web("#3498DB"),
    Color.web("#16A085"),
    Color.web("#F39C12"),
    Color.web("#D35400"),
    Color.web("#C0392B"),
    Color.web("#27AE60"),
    Color.web("#2980B9"),
    Color.web("#2C3E50"),
    Color.web("#E67E22"),
    Color.web("#1ABC9C"),
    Color.web("#9B59B6"),
    Color.web("#2ECC71"),
    Color.web("#34495E"),
    Color.web("#7F8C8D"),
    Color.web("#C2185B"),
    Color.web("#512E5F"),
    Color.web("#154360"),
    Color.web("#0E6251"),
    Color.web("#784212"),
    Color.web("#7B241C"),
    Color.web("#1B4F72"),
    Color.web("#641E16"),
    Color.web("#4A235A"),
    Color.web("#0B5345")
};

public void saveResult(char[][] matrix, int size, String filename)
{
    // remove .txt extension (will be added again)
    int trimmedFileNameIndex = filename.lastIndexOf(".");
    String trimmedFileName = filename.substring(0,
trimmedFileNameIndex);

    String resultFileName = "test/" + trimmedFileName +
"_solved.txt";

    try{
        FileWriter fileWrite = new FileWriter(resultFileName);
        for (int i = 0; i < size; i++) {
```

```

        for (int j = 0; j < size; j++) {
            fileWrite.write(matrix[i][j]);
        }
        fileWrite.write("\n");
    }
    fileWrite.close();

} catch (IOException e) {
    e.printStackTrace();
}

}

public int charToColor (char[][] matrix, int i, int j) {

    char region = matrix[i][j];
    int colorIndex = Character.toUpperCase(region) - 'A';
    colorIndex = colorIndex % 26;
    return colorIndex;

}

public void saveToImage(char[][] matrix, int[][] temp, int size,
String fileName) {
    int cellSize = 100;
    int cellWidth = size * cellSize;
    int cellHeight = size * cellSize;

    Canvas canvas = new Canvas(cellWidth, cellHeight);
    GraphicsContext gc = canvas.getGraphicsContext2D();

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            double x = j * cellSize;
            double y = i * cellSize;
            int colorIndex = charToColor(matrix, i, j);

            // color region
            gc.setFill(colors[colorIndex]);
            gc.fillRect(x, y, cellSize, cellSize);

            //border

```



```

        gc.setStroke(Color.BLACK);
        gc.setLineWidth(2);
        gc.strokeRect(x, y, cellSize, cellSize);

        // queen
        if (temp[i][j] == 1) {
            gc.setFill(Color.GOLD);
            gc.setFont(Font.font("Segoe UI Emoji",
FontWeight.BOLD, cellSize * 0.6));
            gc.setTextAlign(TextAlignment.CENTER);

            gc.fillText("👑", x + (cellSize / 2.0), y +
(cellSize / 1.4));

            gc.setEffect(null);
        }
    }
}
try {
    WritableImage writableImage = new WritableImage(cellWidth,
cellHeight);
    canvas.snapshot(null, writableImage);

    int trimmedFileNameIndex = fileName.lastIndexOf(".");
    String trimmedFileName = fileName.substring(0,
trimmedFileNameIndex);

    File outputFile = new File("test/" + trimmedFileName +
"_solved.png");
    ImageIO.write(SwingFXUtils.fromFXImage(writableImage,
null), "png", outputFile);

} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

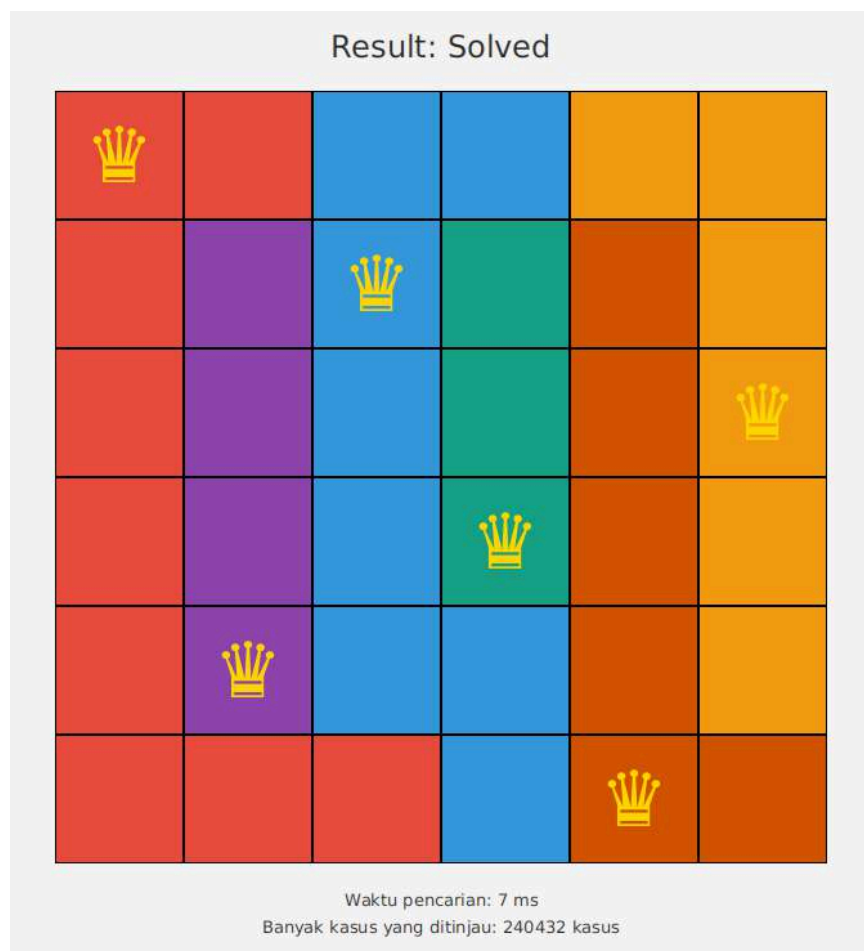
4 Hasil Pengujian

Berikut hasil pengujian yang dilakukan pada berbagai use case dan validasi yang ada pada program ini. Perlu diketahui bahwa test case yang dilakukan terdapat dalam direktori ./data, sedangkan hasil test case yang menemukan solusi akan dimasukkan ke dalam direktori ./test, dalam bentuk .txt maupun .png (image).

4.1 Test Case 1 ($n == 6$ kasus valid)

```
AACCEE  
ABCD FE  
ABCD FE  
ABCD FE  
ABCCFE  
AAACFF
```

Hasil:

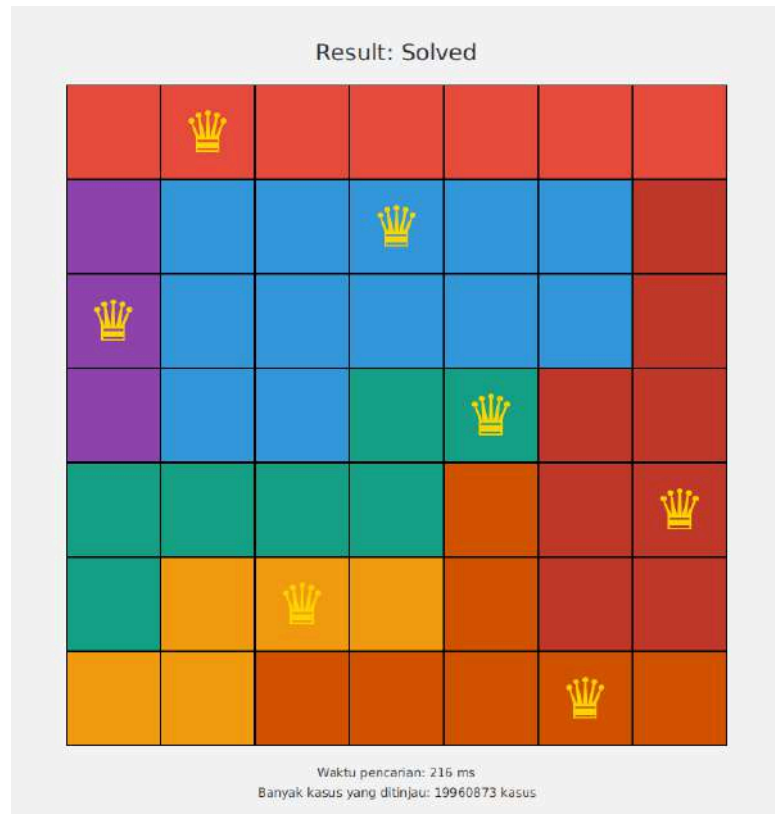


```
#ACCEE  
AB#DFE  
ABCD F#  
ABC#FE  
A#CCFE  
AAAC#F
```

4.2 Test Case 2 ($n == 7$ kasus valid)

AACCEE
ABCD FE
ABCD FE
ABCD FE
ABCCFE
AAACFF

Hasil:

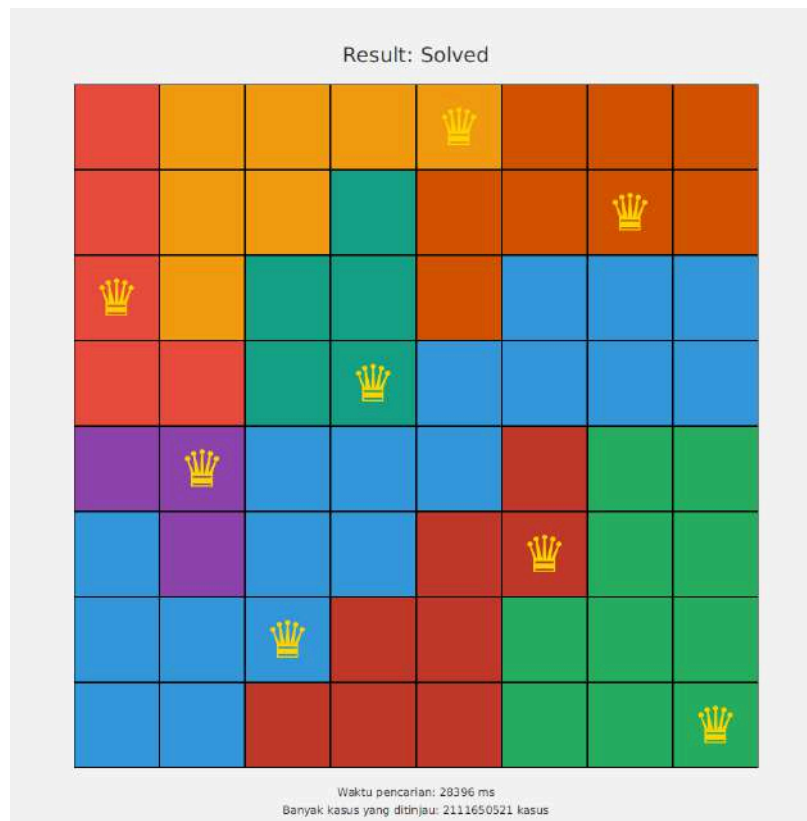


A#AAAAA
BCC#CCG
#CCCCCG
BCCD#GG
DDDDFG#
DE#EFGG
EEFFF#F

4.3 Test Case 3 ($n == 8$ kasus valid)

AACCEE
ABCD FE
ABCD FE
ABCD FE
ABCCFE
AAACFF

Hasil:



AEEE#FFF
AEEDFF#F
#EDDFCCC
AAD#CCCC
B#CCCGHH
CBCCG#HH
CC#GGHHH
CCGGGHH#

4.4 Test Case 4 (rows dan column tidak valid)

AAAA
BBBB
CCCC

Panjang dan lebar tidak simetris

4.5 Test Case 5 (banyaknya region tidak simetris dengan rows dan column size)

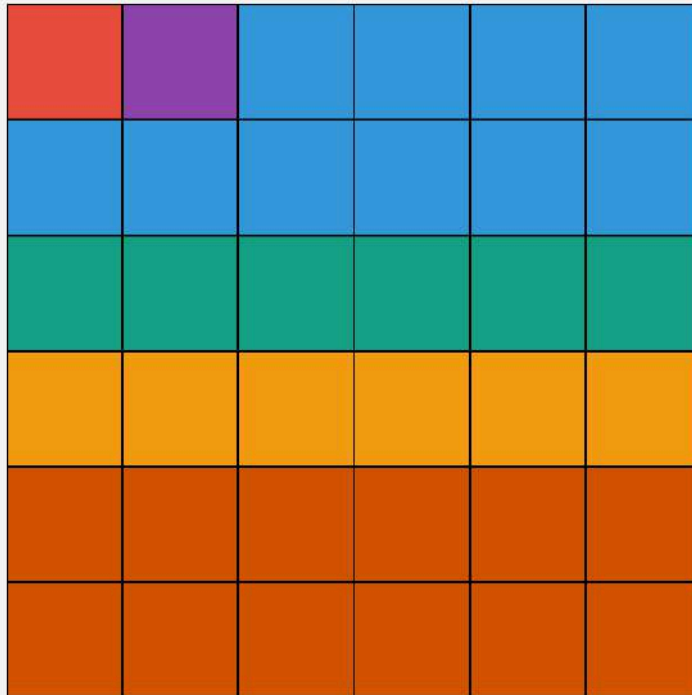
```
AAAAA  
AAAAA  
BBBBB  
BBBBB  
CCCCC
```

Daerah tidak simetris dengan dimensi papan

4.6 Test Case 6 (input board valid namun tidak menemukan solusi)

```
ABCCCC  
CCCCCC  
DDDDDD  
EEEEEE  
FFFFFF  
FFFFFF
```

Result: No Solution Found



Waktu pencarian: 78 ms
Banyak kasus yang ditinjau: 1947792 kasus

4.7 *Test Case 7 (input manual dan mendapatkan solusi yang valid)*

Queens LinkedIn Solver

Select Board Size:

Queens LinkedIn Solver

Enter Regions (A-Z):

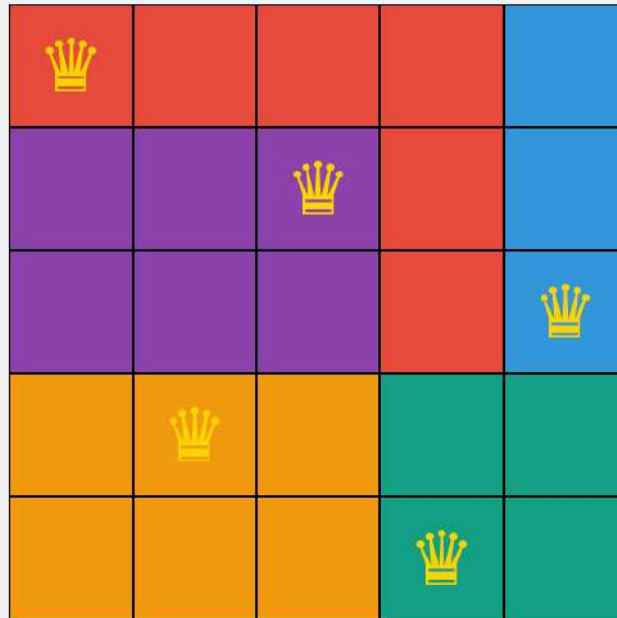
A	A	A	A	C
B	B	B	A	C
B	B	B	A	C
E	E	E	D	D
E	E	E	D	D

Queens LinkedIn Solver

Upload File (.txt)

Manual Input

Result: Solved



Waktu pencarian: 1 ms

Banyak kasus yang ditinjau: 8097 kasus

Save as .TXT

Save as Image

5 Lampiran

- ***Github Repository***

https://github.com/Naufal-Pinasthika/Tucil1_13524013

- ***Tabel Penilaian***

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 srta menyimpan solusi dalam berkas .txt	✓	
5	Program memiliki Graphical User Interface (GUI)	✓	
6	Program dapat menyimpan solusi dalam bentuk file gambar	✓	

6 Pernyataan tidak melakukan kecurangan

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



Anindya Naufal Pinasthika