TUCIL 1 IF2211 STRATEGI ALGORITMA
Penyelesaian Permainan Queens LinkedIn

Disusun oleh:
13524066 - Nathanael Gunawan

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2026

BAB 1
Deskripsi Persoalan dan Algoritma

1.1. Latar Belakang Persoalan

Laporan ini dibuat untuk menyelesaikan tugas kecil pertama mata kuliah IF2211 Strategi Algoritma untuk melengkapkan kode program yang telah dibuat untuk menyelesaikan permainan Queens LinkedIn. Permain Queens LinkedIn adalah permainan logika yang tersedia pada situs jejaring profesional LinkedIn. Tujuan dari permainan ini adalah menempatkan queen pada sebuah papan persegi berwarna sehingga hanya satu queen pada tiap baris, kolom, dan daerah warna. Selain itu, satu queen tidak dapat ditempatkan bersebelahan dengan queen lainnya, termasuk secara diagonal. Pada tugas ini, diharuskan membuat kode program penyelesaian dengan metode *brute force* murni mengikuti materi perkuliahan.

1.2. Permainan Queens LinkedIn

Queens adalah game logika di mana Anda mengisi kotak sehingga hanya ada satu Ratu per baris, kolom, dan wilayah berwarna. Peraturan permainan queens sebagai berikut:

- Setiap baris, kolom, dan wilayah berwarna harus berisi hanya satu simbol Mahkota (Ratu).
- Simbol mahkota tidak dapat ditempatkan di sel yang berdekatan, termasuk secara diagonal.
- Klik atau ketuk sel untuk beralih antara sel kosong, simbol bertanda, dan simbol Mahkota.
- Gunakan simbol yang ditandai untuk menghilangkan sel yang tidak boleh diisi simbol Mahkota.

1.3. Algoritma Brute Force

Algoritma Brute Force adalah algoritma yang digunakan untuk menyelesaikan masalah dengan cara mencoba semua kemungkinan yang ada untuk menemukan kemungkinan yang benar atau memenuhi. Metode ini sering digunakan sebagai basis pembanding metode lain dikarenakan pasti menemukan solusi jika ada. Pada tugas ini digunakan metode brute force murni (Exhaustive Search) tanpa backtracking ataupun optimasi lainnya. Pendekatan yang dilakukan adalah merekursi baris per baris, dengan meng-enumerasi semua penempatan N-ratu (satu per baris) terlebih dahulu kemudian baru divalidasi setiap konfigurasinya. Berbeda dengan backtracking yang baru menempatkan ratu setelah dipastikan tidak menimbulkan konflik.

BAB 2
Penerapan Algoritma

2.1. Langkah Algoritma

- Inisialisasi

  Algoritma dimulai dengan membuat array positions[0..N-1] untuk menyimpan posisi kolom ratu pada setiap baris. Disini juga dimulai pencatatan waktu dan menginisialisasi counter iterasi dengan 0.

- Rekursi Baris

  Fungsi dipanggil mulai dari row = 0. Untuk setiap baris, dicoba semua kemungkinan kolom dari 0 hingga N-1. Nilai positions[row] = col diset, lalu fungsi dipanggil rekursif untuk row+1.

- Kondisi Basis

  Jika row == N, berarti seluruh N baris telah diisi (satu ratu per baris). Dengan ini, telah didapatkan satu konfigurasi lengkap. iterationCount dinaikkan 1.

- Validasi Konfigurasi

  Fungsi isValid(positions) dipanggil. Untuk setiap pasangan ratu (i, j) dengan i < j, diperiksa apakah terjadi konflik melalui conflictsBetween(i, positions[i], j, positions[j]). Tiga kondisi konflik: (a) kolom sama, (b) bertetangga diagonal, (c) warna sel sama.

- Solusi

  Jika isValid mengembalikan true, posisi ratu ditulis ke papan melalui board.placeQueen() dan fungsi mengembalikan true, menghentikan pencarian lebih lanjut.

- Tidak ada solusi pada cabang

  Jika isValid mengembalikan false, fungsi kembali ke pemanggil untuk mencoba kolom berikutnya (backtrack implisit karena loop for di level atas).

- Tidak ada solusi

  Jika semua kombinasi telah dicoba dan tidak ada yang valid, fungsi mengembalikan false ke pemanggil awal dan program menyatakan 'Tidak ada solusi'.

2.2. Pseudocode Solver (Bagian Algoritma Brute Force)

```
SOLVE():
        positions[0..N-1] = array kosong
        RETURN solveRecursive(0)
solveRecursive(row):
        IF row == N THEN
                iterationCount++
                IF isValid(positions) THEN
                        PlaceQueens()
                        RETURN true
```

```
            RETURN false
        FOR col FROM 0 TO N-1:
            positions[row] = col
            IF solveRecursive(row + 1) THEN
                RETURN true
        RETURN false

    isValid(positions):
        FOR i FROM 0 TO N-1:
            FOR j FROM i+1 TO N-1:
                IF conflictsBetween(i, positions[i], j, positions[j])
            THEN
                        RETURN false
        RETURN true

    conflictsBetween(r1, c1, r2, c2):
        IF c1 == c2 : kolom sama -> konflik
        IF |r1-r2|<=1 AND |c1-c2|<=1 : tetangga diagonal -> konflik
        IF cells[r1][c1] == cells[r2][c2] : warna sama -> konflik
        RETURN false
```

2.3. Fungsi Pada Setiap File

File	Paket	Deskripsi
Board.java	tucil1.model	Model papan: menyimpan grid, posisi ratu, dan logika konflik
Solver.java	tucil1.solver	Logika brute force, rekursi
SolverResult.java	tucil1.solver	Hasil solver untuk digunakan GUI
Main.java	tucil1	I/O file dan cli
pom.xml	–	Konfigurasi Maven dan dependensi JavaFX 17

2.3.1. Board.java

2.3.1.1. Atribut

Atribut	Deskripsi

private int size	Ukuran papan (dimensi N x N)
private char[][] cells	Matriks 2D yang menyimpan warna/karakter setiap sel papan
private int[] queenPositions	Array 1D yang menyimpan posisi kolom ratu untuk setiap baris. Nilai -1 berarti baris tersebut belum memiliki ratu

2.3.1.2. Konstruktor

Konstruktor	Deskripsi
public Board(int size, char[][] cells)	Konstruktor yang membuat objek Board baru dengan ukuran dan sel-sel papan yang sudah ditentukan..

2.3.1.3. Metode

Metode	Deskripsi
public boolean conflictsBetween(int row1, int col1, int row2, int col2)	Memeriksa apakah dua ratu saling konflik.
public boolean isSafe(int row, int col)	Memeriksa apakah kolom `col` pada baris `row` aman untuk penempatan ratu.
public void placeQueen(int row, int col)	Menempatkan ratu pada posisi baris `row` dan kolom `col`.
public void removeQueen(int row)	Menghapus ratu dari baris `row’
public void reset()	Menghapus semua ratu dari papan.
public char[][] getCells()	Getter yang mengembalikan referensi ke matriks warna/karakter papan.

public int getSize()	Getter yang mengembalikan ukuran papan (N dari N x N).
public int getQueenPosition(int row)	Mengembalikan posisi kolom ratu pada baris `row`
public void printBoard()	Menampilkan papan ke standar output.
public String getBoardAsString()	Mengubah papan menjadi string representation.

2.3.2. Solver.java

2.3.2.1. Atribut

Atribut	Deskripsi
private Board board	Objek papan yang akan diselesaikan
private long iterationCount	Jumlah iterasi (baris yang telah dievaluasi) selama pencarian
private long startTime	Waktu mulai pencarian dalam milidetik.
private int[] positions	Array sementara yang menyimpan posisi kolom ratu untuk setiap baris selama pencarian.
private int updateFrequency	Kelipatan iterasi yang ditunjukkan untuk *live update*. Dibuat beragam untuk mempercepat perhitungan.

2.3.2.2. Konstruktor

Konstruktor	Deskripsi
public Solver(Board board)	Konstruktor yang membuat objek Solver baru dengan papan yang akan diselesaikan.

2.3.2.3. Metode

Metode	Deskripsi
public SolverResult solve()	Metode utama yang memulai proses pencarian solusi.
private boolean solveRecursive(int row)	Fungsi recursive yang melakukan pencarian.
private boolean isValid(int[] positions)	Memvalidasi apakah konfigurasi ratu pada array `positions` valid.
private void printCurrentState()	Menampilkan state papan saat ini ke console, menunjukkan posisi temporary ratu selama pencarian.
public long getIterationCount()	Mengembalikan jumlah iterasi yang telah dilakukan selama pencarian
public long getElapsedTime()	Mengembalikan waktu yang telah berlalu sejak pencarian dimulai (dalam milidetik)

2.3.3. Main.java

2.3.3.1. Atribut

Tidak ada atribut

2.3.3.2. Konstruktor

Tidak ada konstruktor

2.3.3.3. Metode

Metode	Deskripsi
public static void main(String[] args)	Metode entry point utama aplikasi, menjalankan alur program CLI.
public static Board readBoardFromFile(String filename)	Membaca file text yang berisi papan dan mengkonversinya menjadi objek Board.
private static void validateBoard(Board board)	Memvalidasi bahwa papan memiliki jumlah

	warna/karakter yang sesuai dengan ukuran papan.
public static String saveSolution(Board board, String inputFilename)	Menyimpan solusi ke file baru.

2.3.4. SolverResult.java

2.3.4.1. Atribut

Atribut	Deskripsi
private boolean solved	Menunjukkan apakah solusi berhasil ditemukan (true) atau tidak (false)
private int[] queenPositions	Array yang menyimpan posisi kolom ratu untuk setiap baris dalam solusi (jika ada)
private long iterationCount	Jumlah iterasi (baris yang dievaluasi) yang dilakukan selama pencarian
private long executionTime	Waktu eksekusi pencarian dalam milidetik

2.3.4.2. Konstruktor

Konstruktor	Deskripsi
public SolverResult(boolean solved, int[] queenPositions, long iterationCount, long executionTime)	Konstruktor yang membuat objek `SolverResult` baru dengan hasil pencarian

2.3.4.3. Metode

Metode	Deskripsi
public boolean isSolved()	Mengembalikan status apakah solusi berhasil ditemukan
public int[] getQueenPosition()	Mengembalikan array posisi kolom ratu untuk setiap baris

	dalam solusi
public long getIterationCount()	Mengembalikan jumlah iterasi (baris yang dievaluasi) yang dilakukan selama pencarian.
public long getExecutionTime()	Mengembalikan waktu eksekusi pencarian dalam milidetik

BAB 3
Kode Program dalam Bahasa Java

3.1. Repositori Github
https://github.com/NathanaelGun/Tucil1_13524066

3.2. Board.java

```java
package tucil1.model;

public class Board {
    private int size;
    private char[][] cells;
    private int[] queenPositions;

    public Board(int size, char[][] cells) {
        this.size = size;
        this.cells = cells;
        this.queenPositions = new int[size];

        for (int i = 0; i < size; i++) {
            queenPositions[i] = -1;
        }
    }

    public boolean conflictsBetween(int row1, int col1, int row2, int col2) {
        if (col1 == col2) {
            return true;
        }
        if (Math.abs(row1 - row2) <= 1 && Math.abs(col1 - col2) <= 1) {
            return true;
        }
        if (cells[row1][col1] == cells[row2][col2]) {
            return true;
        }
        return false;
    }

    public boolean isSafe(int row, int col) {
        for (int i = 0; i < row; i++) {
            if (conflictsBetween(row, col, i, queenPositions[i])) {
                return false;
            }
        }
        return true;
    }

    public void placeQueen(int row, int col) {
```

```
        queenPositions[row] = col;
    }

    public void removeQueen(int row) {
        queenPositions[row] = -1;
    }

    public void reset() {
        for (int i = 0; i < getSize(); i++) {
            removeQueen(i);
        }
    }

    public char[][] getCells() {
        return cells;
    }

    public int getSize() {
        return size;
    }

    public int getQueenPosition(int row) {
        return queenPositions[row];
    }

    public void printBoard() {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (queenPositions[i] == j) {
                    System.out.print(cells[i][j] + "# ");
                } else {
                    System.out.print(cells[i][j] + "  ");
                }
            }
            System.out.println();
        }
    }

    public String getBoardAsString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (queenPositions[i] == j) {
                    sb.append('#');
                } else {
                    sb.append(cells[i][j]);
                }
            }
```

```
            sb.append('\n');
        }
        return sb.toString();
    }

}
```

3.3. Solver.java

```
package tucil1.solver;

import tucil1.model.Board;

public class Solver {
    private Board board;
    private long iterationCount = 0;
    private long startTime;
    private int[] positions;
    private int updateFrequency;

    public Solver(Board board) {
        this.board = board;
        this.updateFrequency = getUpdateFrequency(board.getSize());
    }

    private int getUpdateFrequency(int size) {
        if (size <= 6) return 100;
        if (size == 7) return 10000;
        if (size == 8) return 100000;
        return 1000000;
    }



    public SolverResult solve() {
        startTime = System.currentTimeMillis();
        positions = new int[board.getSize()];
        boolean solved = solveRecursive(0);
        long elapsed = System.currentTimeMillis() - startTime;
        return new SolverResult(solved, positions.clone(),
iterationCount, elapsed);
    }
```

```
    private boolean solveRecursive(int row) {
        if (row == board.getSize()) {
            iterationCount++;

            if (iterationCount % updateFrequency == 0) {
                printCurrentState();
            }

            if (isValid(positions)) {
                for (int i = 0; i < board.getSize(); i++) {
                    board.placeQueen(i, positions[i]);
                }
                return true;
            }
            return false;
        }

        for (int col = 0; col < board.getSize(); col++) {
            positions[row] = col;
            if (solveRecursive(row + 1)) {
                return true;
            }
        }
        return false;
    }

    private boolean isValid(int[] positions) {
        for (int i = 0; i < positions.length; i++) {
            for (int j = i + 1; j < positions.length; j++) {
                if (board.conflictsBetween(i, positions[i], j,
positions[j])) {
                    return false;
                }
            }
        }
        return true;
    }

    private void printCurrentState() {
        System.out.println("\n--- Iterasi ke-" + iterationCount + "
---");
```

```
        for (int i = 0; i < board.getSize(); i++) {
            for (int j = 0; j < board.getSize(); j++) {
                if (positions[i] == j) {
                    System.out.print("[#]");
                } else {
                    System.out.print("[" + board.getCells()[i][j] +
"]");
                }
            }
            System.out.println();
        }
        System.out.flush();
    }

    public long getIterationCount() {
        return iterationCount;
    }

    public long getElapsedTime() {
        return System.currentTimeMillis() - startTime;
    }
}
```

3.4. SolverResult.java

```
package tucil1.solver;

public class SolverResult {
    private boolean solved;
    private int[] queenPositions;
    private long iterationCount;
    private long executionTime;


    public SolverResult(boolean solved, int[] queenPositions, long iterationCount,
long executionTime) {
        this.solved = solved;
        this.queenPositions = queenPositions;
        this.iterationCount = iterationCount;
        this.executionTime = executionTime;
    }

    public boolean isSolved() {
        return solved;
```

```
    }

    public int[] getQueenPosition() {
        return queenPositions;
    }

    public long getIterationCount() {
        return iterationCount;
    }

    public long getExecutionTime() {
        return executionTime;
    }
}
```

3.5. Main.java

```
package tucil1;

import java.io.*;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

import tucil1.model.Board;
import tucil1.solver.Solver;
import tucil1.solver.SolverResult;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("================================");
        System.out.println("         Queens Solver          ");
        System.out.println("================================");
        System.out.print("Masukkan nama file test case: ");
        String filename = scanner.nextLine().trim();

        try {
            Board board = readBoardFromFile(filename);
            validateBoard(board);

            System.out.println("\nPapan awal:");
            board.printBoard();
            System.out.println();
```

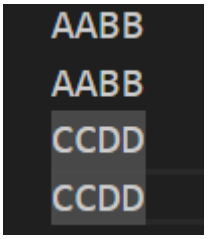
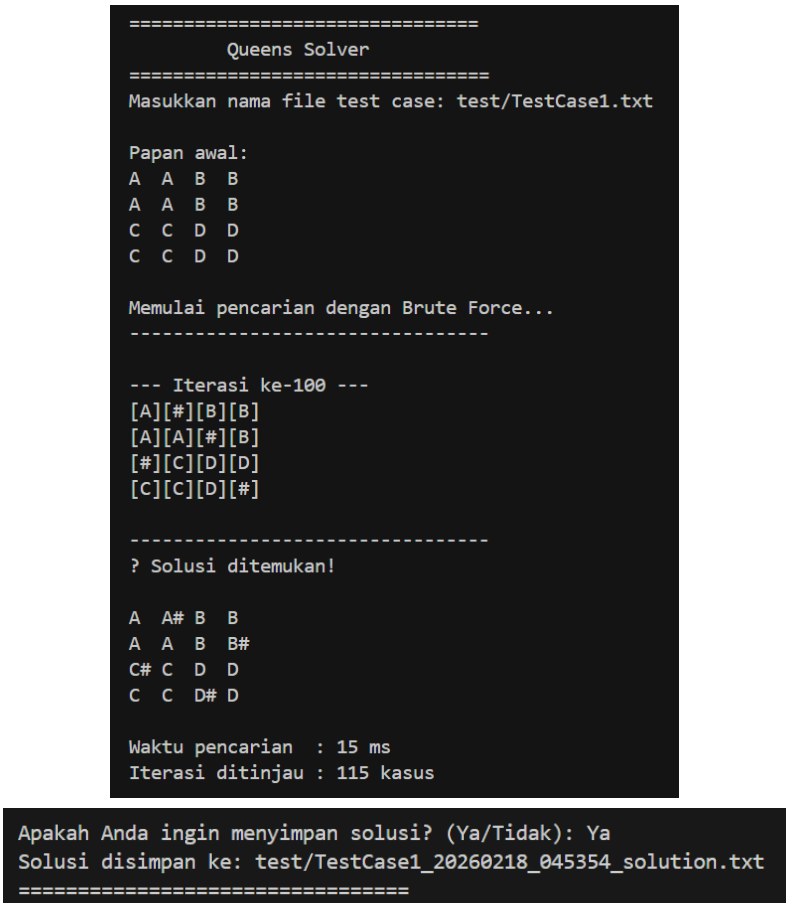
```
            Solver solver = new Solver(board);
            System.out.println("Memulai pencarian dengan Brute Force...");
            System.out.println("--------------------------------");

            SolverResult result = solver.solve();

            System.out.println();
            System.out.println("--------------------------------");

            if (result.isSolved()) {
                System.out.println("✓ Solusi ditemukan!\n");
                board.printBoard();
                System.out.println();
                                    System.out.println("Waktu  pencarian    :  "  +
result.getExecutionTime() + " ms");
                                     System.out.println("Iterasi  ditinjau  :  "  +
result.getIterationCount() + " kasus");

                 System.out.print("\nApakah Anda ingin menyimpan solusi? (Ya/Tidak):
");
                String answer = scanner.nextLine().trim();
                if (answer.equalsIgnoreCase("Ya")) {
                    String savedPath = saveSolution(board, filename);
                    System.out.println("Solusi disimpan ke: " + savedPath);
                }
            }
            else {
                System.out.println("Tidak ada solusi!\n");
                                    System.out.println("Waktu  pencarian    :  "  +
result.getExecutionTime() + " ms");
                                     System.out.println("Iterasi  ditinjau  :  "  +
result.getIterationCount() + " kasus");
            }

        } catch (IOException e) {
            System.out.println("Error membaca file: " + e.getMessage());
        } catch (IllegalArgumentException e) {
            System.out.println("Input tidak valid: " + e.getMessage());
        }

        System.out.println("================================");
        scanner.close();
    }

    public static Board readBoardFromFile(String filename) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(filename));
        StringBuilder content = new StringBuilder();
        String line;
```

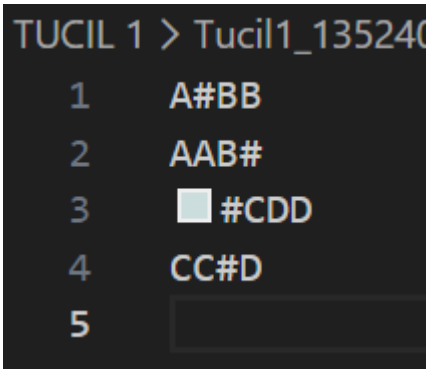
```
        while ((line = reader.readLine()) != null) {
            if (!line.trim().isEmpty()) {
                content.append(line.trim()).append("\n");
            }
        }
        reader.close();

        String[] lines = content.toString().trim().split("\n");
        int size = lines.length;

        for (int i = 0; i < size; i++) {
            if (lines[i].length() != size) {
                throw new IllegalArgumentException(
                    "Papan tidak persegi! Baris " + (i+1) +
                    " punya " + lines[i].length() +
                    " kolom, harusnya " + size
                );
            }
        }

        char[][] cells = new char[size][size];
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                cells[i][j] = lines[i].charAt(j);
            }
        }

        return new Board(size, cells);
    }

    private static void validateBoard(Board board) {
        Set<Character> colors = new HashSet<>();
        char[][] cells = board.getCells();

        for (int i = 0; i < board.getSize(); i++) {
            for (int j = 0; j < board.getSize(); j++) {
                colors.add(cells[i][j]);
            }
        }

        if (colors.size() != board.getSize()) {
            throw new IllegalArgumentException(
                "Jumlah warna (" + colors.size() +
                ") harus sama dengan ukuran papan (" + board.getSize() + ")"
            );
        }
    }
```

```
    public static String saveSolution(Board board, String inputFilename) {
        LocalDateTime now = LocalDateTime.now();
                                            DateTimeFormatter      formatter      =
DateTimeFormatter.ofPattern("yyyyMMdd_HHmmss");
        String timestamp = now.format(formatter);

           String outputFilename = inputFilename.replace(".txt", "_" + timestamp +
"_solution.txt");
        try (PrintWriter writer = new PrintWriter(outputFilename)) {
            writer.print(board.getBoardAsString());
        } catch (IOException e) {
            System.out.println("Error menyimpan file: " + e.getMessage());
        }
        return outputFilename;
    }
}
```

BAB 4
Hasil Pengujian

4.1. Test Case 1

Isi file txt Test Case 1

Hasil CLI Test Case 1

Hasil file Output Test Case 1



(karena terdapat # di depan huruf jadi terlihat seperti kode warna)

4.2. Test Case 2

Isi file txt Test Case 2

Hasil CLI Test Case 2

```
================================
        Queens Solver
================================
Masukkan nama file test case: test/TestCase2.txt

Papan awal:
A  A  B  B  C
A  A  B  B  C
D  D  E  E  C
D  D  E  E  C
D  D  E  E  E

Memulai pencarian dengan Brute Force...
--------------------------------

--- Iterasi ke-100 ---
[#][A][B][B][C]
[#][A][B][B][C]
[D][D][E][#][C]
[D][D][E][E][#]
[D][D][E][E][#]

--- Iterasi ke-200 ---
[#][A][B][B][C]
[A][#][B][B][C]
[D][D][#][E][C]
[D][D][E][E][#]
[D][D][E][E][#]
```

```
--- Iterasi ke-300 ---
[#][A][B][B][C]
[A][A][#][B][C]
[D][#][E][E][C]
[D][D][E][E][#]
[D][D][E][E][#]

--------------------------------
? Solusi ditemukan!

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

Waktu pencarian  : 20 ms
Iterasi ditinjau : 359 kasus

Apakah Anda ingin menyimpan solusi? (Ya/Tidak): Ya
Solusi disimpan ke: test/TestCase2_20260218_045626_solution.txt
================================
```

Hasil file Output Test Case 2

```
#ABBC
AA#BC
DDEE#
D#EEC
DDE#E
```

4.3. Test Case 3

Isi file txt Test Case 3

```
AAABBC
AAABBC
DDDBBC
EEFFGG
EEFFGG
EEFFGG
```

Hasil CLI Test Case 3

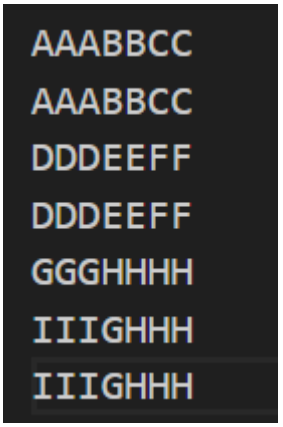
```
================================
        Queens Solver
================================
Masukkan nama file test case: test/TestCase3.txt
Input tidak valid: Jumlah warna (7) harus sama dengan ukuran papan (6)
================================
```

Hasil file Output Test Case 3

Tidak dihasilkan karena input tidak valid

4.4. Test Case 4

Isi file txt Test Case 4



Hasil CLI Test Case 4

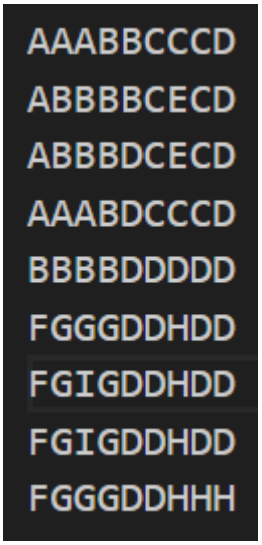
```
================================
        Queens Solver
================================
Masukkan nama file test case: test/TestCase4.txt
Input tidak valid: Jumlah warna (9) harus sama dengan ukuran papan (7)
================================
```

Hasil file Output Test Case 4

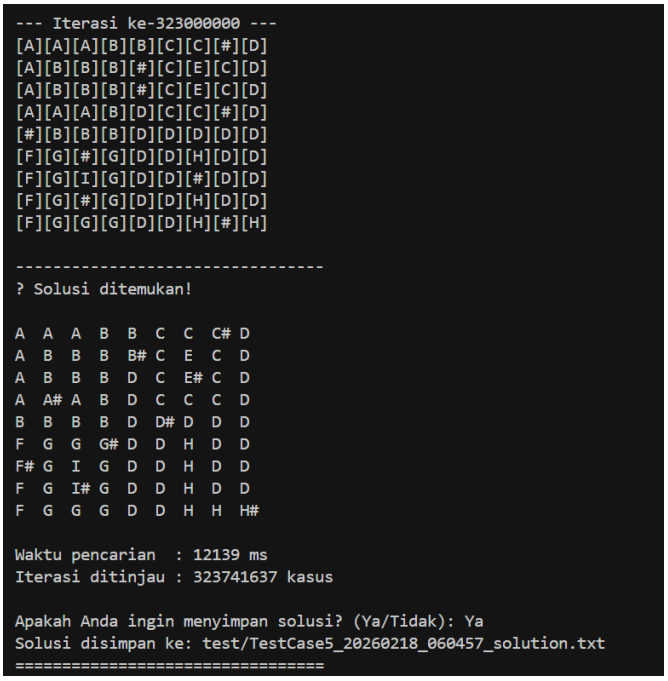
Tidak ada karena input tidak valid

4.5. Test Case 5

Isi file txt Test Case 5

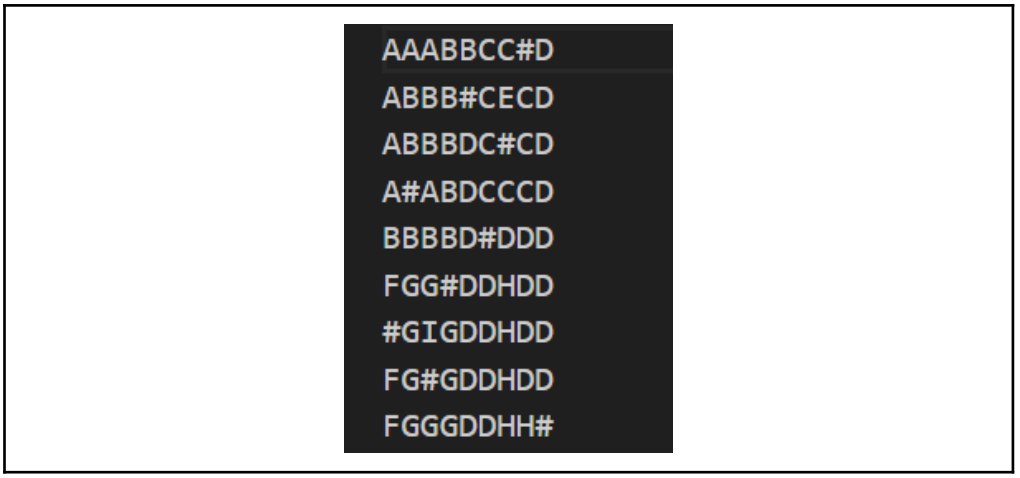


Hasil CLI Test Case 5



Beberapa iterasi terakhir yang dapat terlihat

Hasil file Output Test Case 5


AAABBCC#D
ABBB#CECD
ABBBDC#CD
A#ABDCCCD
BBBBD#DDD
FGG#DDHDD
#GIGDDHDD
FG#GDDHDD
FGGGDDHH#

BAB 5
Lampiran

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

Nathanael Gunawan

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