IF2211 STRATEGI ALGORITMA
LAPORAN AKHIR TUGAS KECIL 1

PENYELESAIAN PERMAINAN QUEENS LINKEDIN

Disusun oleh:

13524093 - Reinsen Silitonga

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG

2026

A. Algoritma Brute Force

a. Langkah-langkah

1. Buat kombinasi awal [0,1,2,...,N-1] yang merepresentasikan N posisi queens pertama pada board N×N.
    a. Contoh untuk N=4:
        i. Kombinasi awal: [0, 1, 2, 3]
        ii. Artinya tempatkan queens di:
        - Sel 0 → baris 0, kolom 0
        - Sel 1 → baris 0, kolom 1
        - Sel 2 → baris 0, kolom 2
        - Sel 3 → baris 0, kolom 3
2. Konversi kombinasi menjadi board. Untuk setiap index dalam kombinasi, hitung koordinat baris dan kolom, lalu tempatkan queens pada posisi tersebut.
    a. Contoh untuk kombinasi [0, 5, 10, 15]:
    - Sel 0: baris = 0/4 = 0, kolom = 0%4 = 0 → board[0][0] = queens
    - Sel 5: baris = 5/4 = 1, kolom = 5%4 = 1 → board[1][1] = queens
    - Sel 10: baris = 10/4 = 2, kolom = 10%4 = 2 → board[2][2] = queens
    - Sel 15: baris = 15/4 = 3, kolom = 15%4 = 3 → board[3][3] = queens
3. Periksa apakah penempatan ratu valid dengan mengecek 5 constraint:
4. Jika valid, simpan sebagai solusi dan selesai.
5. Jika tidak valid, generate kombinasi berikutnya:
    a. Cari posisi paling kanan yang bisa di-increment
    b. Increment posisi tersebut
    c. Reset semua posisi setelahnya menjadi berurutan
    d. Contoh:
    - Kombinasi [0, 1, 2, 3] → tidak valid
    - Posisi 3 bisa increment: 3 → 4
    - Hasil: [0, 1, 2, 4]
    - Posisi 3 increment sampai kombinasi [0, 1, 2, 15] → tidak valid (posisi 3 mentok)
    - Posisi 2 bisa naik: 2 → 3
    - Reset posisi 3: 3+1 = 4
    - Hasil: [0, 1, 3, 4]
    - dan seterusnya
6. Ulangi langkah 2-5 hingga solusi ditemukan atau semua kombinasi habis.
    a. Urutan kombinasi yang dicoba (contoh N=4):
    [0,1,2,3] → [0,1,2,4] → [0,1,2,5] → ... → [0,1,2,15]
    → [0,1,3,4] → [0,1,3,5] → ... → [12,13,14,15]

b. Pseudocode

i. procedure solve

```
procedure solve(input size : integer,
                output solution : MatrixBoolean,
                output found : boolean)
```

```
{ I.S. size > 0 }
{ F.S. found = true jika solusi ditemukan, solution berisi board
solusi
        found = false jika tidak ada solusi }

KAMUS LOKAL
    totalCells : integer
    comb : array [0..size-1] of integer
    queens : MatrixBoolean
    iterationCount : integer
    i, pos, row, col : integer
    lanjut : boolean

ALGORITMA
    startTime ← currentTimeMillis()
    iterationCount ← 0
    solution ← NIL
    found ← false

    totalCells ← size * size

    { inisialisasi kombinasi awal }
    i ← 0
    while i < size do
        comb[i] ← i
        i ← i + 1

    lanjut ← true

    while lanjut = true do
        iterationCount ← iterationCount + 1

        { inisialisasi papan kosong }
        createMatrix(queens, size, size, false)

        { generate board dari kombinasi }
        i ← 0
        while i < size do
            pos ← comb[i]
            row ← pos div size
            col ← pos mod size
            queens[row][col] ← true
            i ← i + 1

        { cek validitas solusi }
        if isValidSolution(queens) then
            solution ← copyBoard(queens)
            found ← true
            lanjut ← false
        else
            { generate kombinasi berikutnya }
```

```
            if not bruteCombination(comb, totalCells, size) then
                found ← false
                lanjut ← false
```

ii. function bruteCombination

```
function bruteCombination(input/output comb : array of integer,
                          input n, k : integer) → boolean
{ I.S. comb adalah kombinasi valid ukuran k dari [0..n-1] }
{ F.S. comb menjadi kombinasi berikutnya jika ada (true),
       jika tidak ada kombinasi berikutnya maka false }

KAMUS LOKAL
    i, j : integer

ALGORITMA
    i ← k - 1
    while (i ≥ 0) and (comb[i] = n - k + i) do
        i ← i - 1

    if i < 0 then
        → false
    else
        comb[i] ← comb[i] + 1
        j ← i + 1
        while j < k do
            comb[j] ← comb[j - 1] + 1
            j ← j + 1
        → true
```

iii. function isValidSolution

```
function isValidSolution(input queens : Board) → boolean
{ I.S. queens terdefinisi }
{ F.S. true jika semua constraint terpenuhi, false jika tidak }

KAMUS LOKAL
    valid : boolean

ALGORITMA
    valid ← true

    if not checkRowConstraint(queens) then
        valid ← false
    else if not checkColConstraint(queens) then
        valid ← false
    else if not checkColorConstraint(queens) then
        valid ← false
    else if not checkNeighborConstraint(queens) then
        valid ← false

    → valid
```

c. **Kompleksitas**
    i. Kompleksitas Waktu: O(C(N^2, N))
    - contoh:
        - Untuk N=4: C(16, 4) = 1,820
        - Untuk N=8: C(64, 8) = 4,426,165,368

B. Source Code Program Java Keseluruhan

a. **QueensBrute**
    i. Deskripsi: Implementasi algoritma Brute Force
    ii. Atribut Utama:
        1. board: char[][] -> Board dengan warna
        2. size: int -> Ukuran board (N)
        3. colorPosition: ColorMap -> Map warna ke posisi
        4. solution: boolean[][] -> Solusi yang ditemukan
        5. iterationCount: long -> Counter iterasi
    iii. Method Utama:
        1. solve() → boolean
            a. Return: true jika solusi ditemukan
            b. Menjalankan algoritma Zero-Pruning hingga menemukan solusi
        2. bruteCombination(int[] comb, int n, int k) → boolean
            a. Parameter:
                i. comb: kombinasi saat ini
                ii. n: total sel (N^2)
                iii. k: jumlah ratu (N)
            b. Return: true jika masih ada kombinasi berikutnya
        3. isValidSolution(boolean[][] queens) → boolean
            a. Parameter: queens – penempatan ratu
            b. Return: true jika memenuhi semua constraint

b. **QueensOptimation**
    i. Deskripsi: Implementasi algoritma Brute Force dengan membatasi tiap queen tidak akan pindah baris agar lebih optimal
    ii. Atribut dan method mirip dengan QueensBrute

c. **PositionList**
    i. Deskripsi: Custom ArrayList untuk menyimpan posisi (row, col)
    ii. Atribut Utama:
        1. data: int[][] → Array 2D untuk menyimpan koordinat
        2. size: int → Jumlah elemen
        3. capacity: int → Kapasitas array
    iii. Method Utama:
        1. add(int row, int col) → void
            a. Parameter:
                i. baris
                ii. kolom

b. Menambahkan posisi ke list
2. get(int index) → int[]
    a. Parameter: index posisi
    b. Return: array [row, col]
3. size() → int
    a. Return: jumlah elemen dalam list
4. resize() → void (private)
    a. Double kapasitas array saat penuh

d. ColorMap

i. Deskripsi: Custom HashMap untuk map warna → list posisi
ii. Atribut Utama:
1. data: MapEntry[] → Array hash table
2. capacity: int → Ukuran hash table
3. size: int → Jumlah entry
iii. Method Utama:
1. put(char key, PositionList value) → void
    a. Parameter:
        i. key: warna (A–Z)
        ii. value: list posisi dengan warna tersebut
    b. Simpan mapping warna ke posisi
2. get(char key) → PositionList
    a. Parameter: key – warna
    b. Return: list posisi dengan warna tersebut, atau null
3. getOrCreate(char key) → PositionList
    a. Parameter: key – warna
    b. Return: list posisi (create baru jika belum ada)
4. hash(char key) → int (private)
    a. Parameter: key – warna
    b. Return: hash index
5. Hash function sederhana: key % capacity

e. IOFiles

i. Deskripsi: Utility untuk input/output file dan parsing
ii. Method Utama:
1. parseBoard(String input) → char[][]
    a. Parameter: input – string board
    b. Return: board 2D
    c. Parse string menjadi board 2D
2. loadTxt(String filename) → char[][]
    a. Parameter: filename – path file
    b. Return: board 2D
    c. Load board dari file .txt
3. saveFile(String filename, String content) → void
    a. Parameter:
    b. filename: path output

c. content: isi file
d. Simpan content ke file

4. getBoardString(char[][] board, boolean[][] queens, int size) → String
    a. Parameter:
        i. board: board dengan warna
        ii. queens: posisi ratu (null jika tidak ada)
        iii. size: ukuran board
    b. Return: string representasi board
    c. Konversi board ke string

f. ImageProcessor

i. Deskripsi: Generate gambar board dengan warna dan crown overlay
ii. Atribut Utama:
    1. LETTER_COLORS: Color[] → 26 warna untuk A–Z
    2. crownImage: Image → Crown overlay untuk queen
iii. Method Utama:
    1. boardToImage(char[][] board, boolean[][] queens) → WritableImage
        a. Parameter:
            i. board: board dengan warna
            ii. queens: posisi ratu (null jika tidak ada)
        b. Return: JavaFX WritableImage
        c. Generate gambar board dengan:
            i. Warna sesuai huruf (A=merah, B=cyan, dst)
            ii. Crown overlay pada posisi queen
            iii. Grid lines untuk pemisah sel
    2. getColorForLetter(char letter) → Color (private)
        a. Parameter: letter (A–Z)
        b. Return: Color untuk huruf tersebut
    3. exportToPNG(WritableImage image, String filename) → void
        a. Parameter:
            i. image: gambar JavaFX
            ii. filename: path output .png
        b. Export WritableImage ke file PNG

g. QueensMain

i. Deskripsi: Main window dengan tab untuk Text dan Image mode dan mengatur layout utama aplikasi

h. QueensView

i. Deskripsi: Tab untuk mode text (input manual, solve, live update) dan menampilkan board sebagai text dengan live iteration counter

i. QueensImage

i. Deskripsi: Tab untuk mode image (load file, solve, export PNG) dan menampilkan board sebagai gambar berwarna dengan crown

j. QueensController

i. Deskripsi: Controller untuk QueensView (text mode)

ii. Handle user input, solve, dan display update

k. QueensImageController

i. Deskripsi: Controller untuk QueensImage (image mode)

ii. Handle load file, solve, export PNG

l. QueensSolverApp

i. Deskripsi: Entry point aplikasi

ii. Extends JavaFX Application

iii. Method:

1. start(Stage primaryStage) → void: Launch QueensMain window
2. main(String[] args) → void: Entry point program

C. Source Code Algoritma Brute Force

a. Solve

```
public boolean solve() {
     startTime = System.currentTimeMillis();
     iterationCount = 0;
     solution = null;

     int totalCells = size * size;
     int[] comb = new int[size];
     for (int i = 0; i < size; i++) {
         comb[i] = i;
     }
     while (true) {
         iterationCount++;
         boolean[][] queens = new boolean[size][size];
         for (int i = 0; i < size; i++) {
             int pos = comb[i];
             int row = pos / size;
             int col = pos % size;
             queens[row][col] = true;
         }

         // buat visualisasi
         if (visualize && iterationCount % visualizationInterval == 0) {
             String boardStr = IOfiles.getBoardString(board, queens, size);
             String message = "Iteration " + iterationCount + "\n\n" + boardStr;
             javafx.application.Platform.runLater(() -> {
                 currentStateProperty.set(message);
             });
             try {
                 Thread.sleep(50); // 50ms delay
             } catch (InterruptedException e) {
                 Thread.currentThread().interrupt();
             }
         }

         if (isValidSolution(queens)) {
             solution = copyBoard(queens);
             return true; // solusi ketemu langsung break
         }
```

```
            // generate kombinasi berikutnya
            if (!bruteCombination(comb, totalCells, size)) {
                break;
            }
        }
        return false; // tidak ada solusi
    }
```

b. bruteCombination

```
    public boolean bruteCombination(int[] comb, int n, int k) {
        int i = k - 1;
        while (i ≥ 0 && comb[i] == n - k + i) {
            i--;
        }
        if (i < 0) {
            return false;
        }

        comb[i]++;
        for (int j = i + 1; j < k; j++) {
            comb[j] = comb[j - 1] + 1;
        }
        return true;
    }
```

c. isValidSolution

```
    public boolean isValidSolution(boolean[][] queens) {
        int totalQueens = 0;
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (queens[i][j]) totalQueens++;
            }
        }
        if (totalQueens ≠ size) return false;

        // check row
        for (int i = 0; i < size; i++) {
            int count = 0;
            for (int j = 0; j < size; j++) {
                if (queens[i][j]) count++;
            }
            if (count ≠ 1) return false;
        }

        // check column
        for (int j = 0; j < size; j++) {
            int count = 0;
            for (int i = 0; i < size; i++) {
                if (queens[i][j]) count++;
            }
            if (count ≠ 1) return false;
        }

        // check color
        char[] colors = colorPosition.keySet();
        for (int c = 0; c < colors.length; c++) {
            char color = colors[c];
            PositionList positions = colorPosition.get(color);
```

```
            int count = 0;
            for (int p = 0; p < positions.size(); p++) {
                int[] pos = positions.get(p);
                if (queens[pos[0]][pos[1]]) count++;
            }
            if (count ≠ 1) return false;
        }

        // check neighbor
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (queens[i][j]) {
                    for (int dr = -1; dr ≤ 1; dr++) {
                        for (int dc = -1; dc ≤ 1; dc++) {
                            if (dr == 0 && dc == 0) continue;
                            int nr = i + dr;
                            int nc = j + dc;
                            if (nr ≥ 0 && nr < size && nc ≥ 0 && nc < size) {
                                if (queens[nr][nc]) {
                                    return false;
                                }
                            }
                        }
                    }
                }
            }
        }

        return true;
    }
```

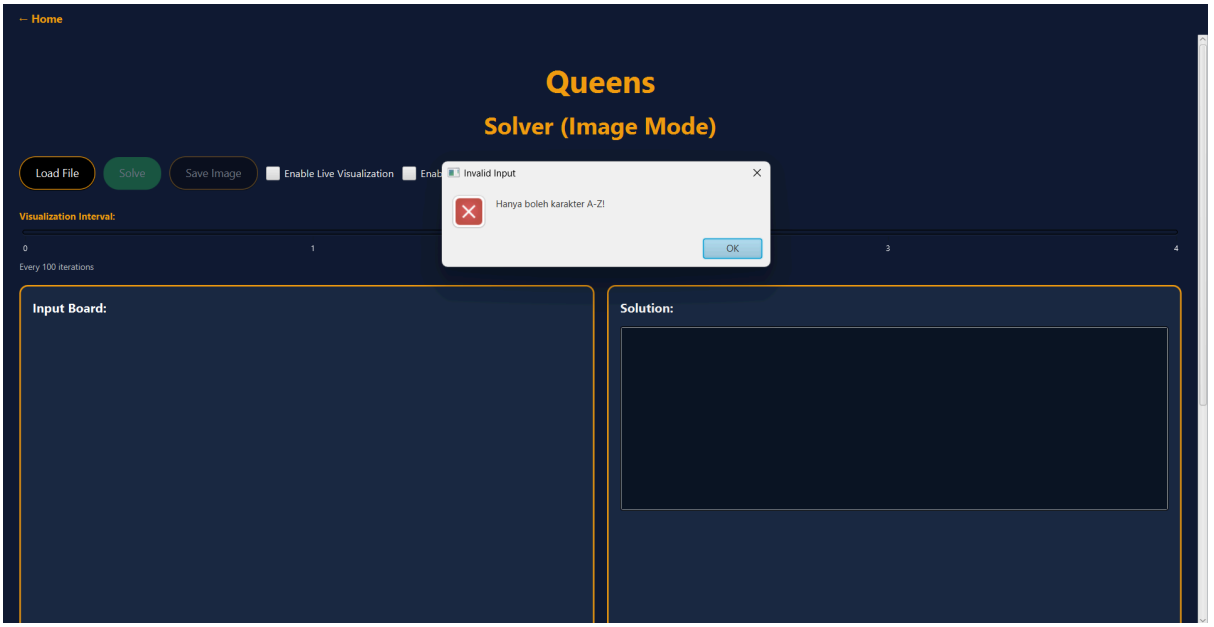
D. Screenshot Test Case

1. Test Case 7x7 Valid Board (Text)
   Input & Output (Live Update tiap 1000 iterasi):

2. Test Case 4x4 Valid Board (Input Text, Output Image)

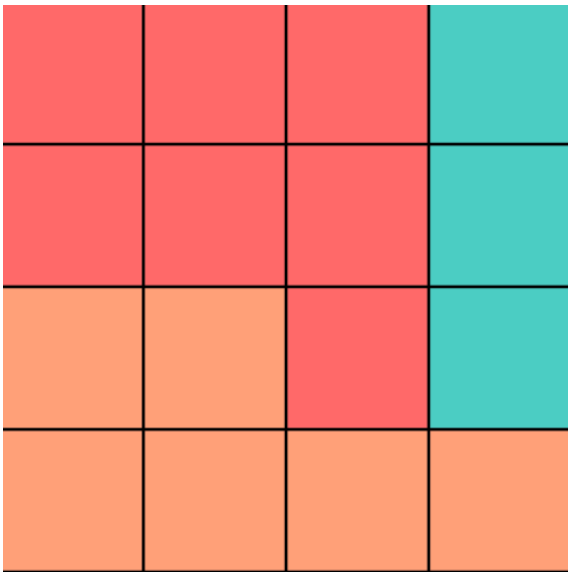
Input & Output (Live Update tiap 10 iterasi):

3. Test Case Invalid Board (Ada angka di input)
   Input:
   A4AB
   AAAB
   DDAB
   DCCC
   Output:

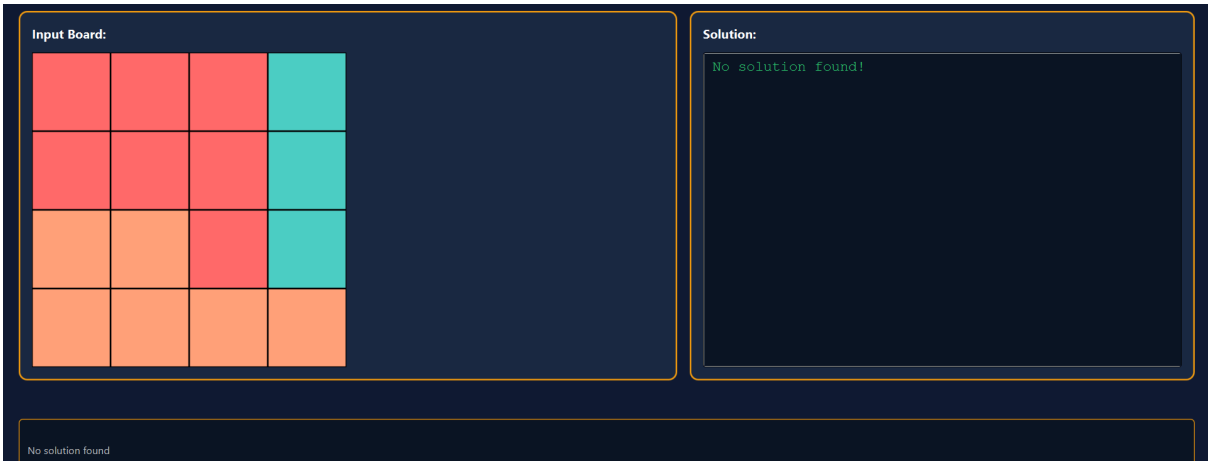


4. Test Case No Solution

Input (txt):

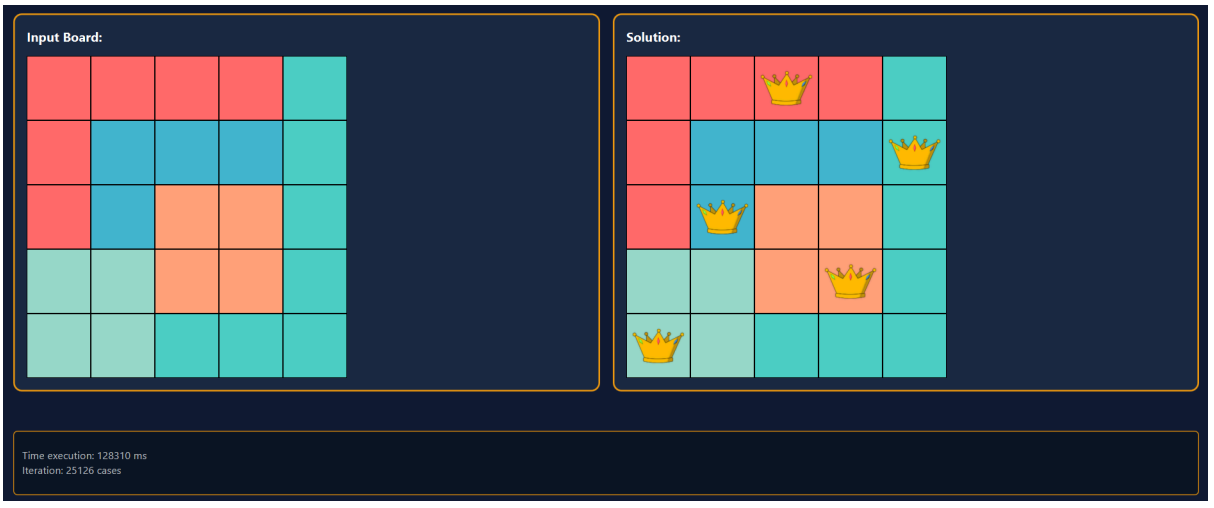


Output:



5. Test Case 5x5

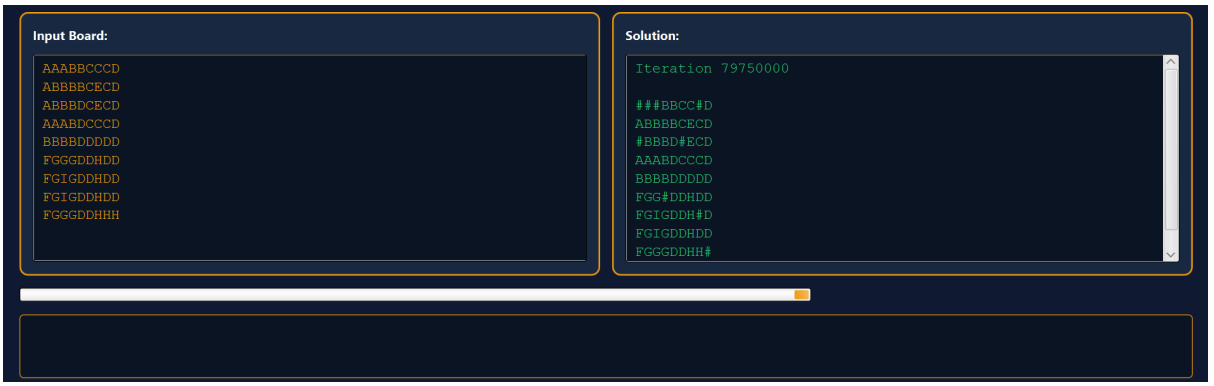
Input & Output (Live Update tiap 10 iterasi):



6. Test Case 6x6

Input & Output (Live Update tiap 100 iterasi):

7. Test Case 9x9 (terlalu lama, tidak selesai)
   Input & Output (Live Update tiap 10000 iterasi):



E. Pranala Repository

- https://github.com/re1nsilitonga/Tucil1_13524093

F. Lampiran

a. Pernyataan tidak melakukan kecurangan

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

Reinsen Silitonga

b. Tabel poin yang dikerjakan

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	✓	
7	Program dapat membaca masukan file gambar		✓