Laporan Tugas Kecil 1

IF2211 Strategi Algoritma

Penyelesaian Permainan Queens Linkedin

Semester II Tahun 2025/2026

Narendra Dharma Wistara M.
13524044

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

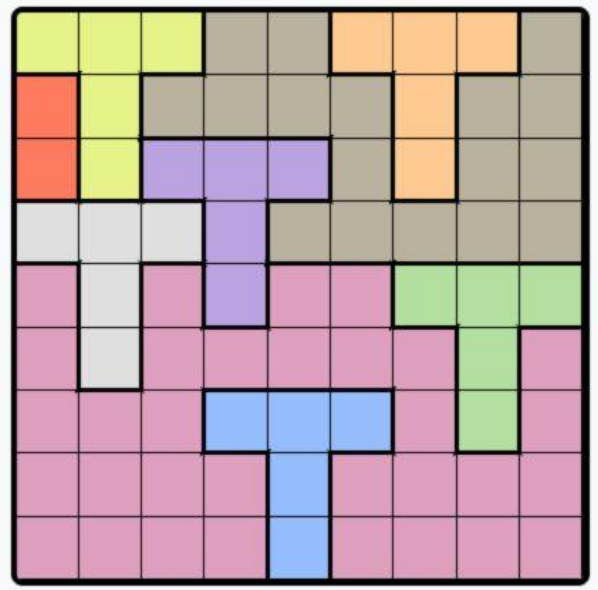
Daftar Isi

1 Deskripsi Penugasan

Queens adalah gim logika yang tersedia pada situs jejaring profesional LinkedIn. Tujuan dari gim ini adalah menempatkan *queen* pada sebuah papan persegi berwarna sehingga terdapat hanya satu *queen* pada tiap baris, kolom, dan daerah warna. Selain itu, satu *queen* tidak dapat ditempatkan bersebelahan dengan *queen* lainnya, termasuk secara diagonal.

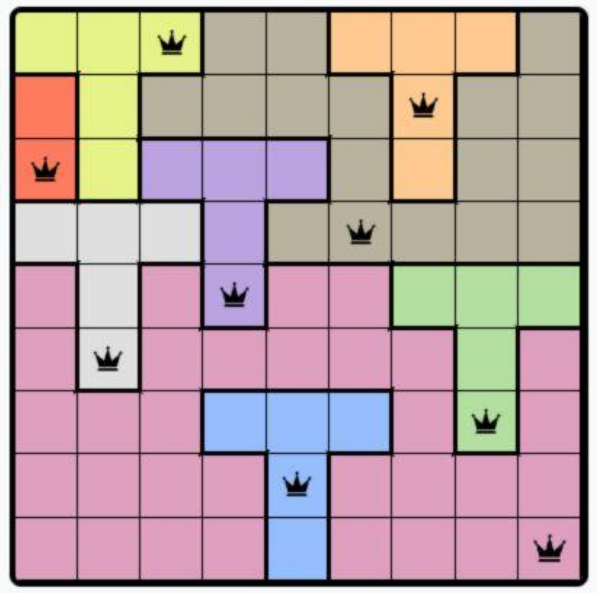
Tugas anda adalah membuat program yang dapat menemukan satu solusi penempatan *queen* pada suatu papan berwarna yang diberkan atau menmpilkan bahwa tidak ada solusi yang valid. Program melakukan solusi menggunakan algoritma brute force.

Misal, diberikan papan sebagai berikut. Untuk tugas ini, papan selalu dimulai kosong.



Gambar 1: Papan Permainan Queens

Di bawah ini satu-satunya solusi valid. Perhatikan bahwa tiap baris, kolom, dan daerahj warna sudah memiliki satu ditempati satu *queen*.



Gambar 2: Solusi Permainan Queen

2 Spesifikasi Tugas

2.1 Spesifikasi Wajib

- Buatlah sebuah program sederhana dalam bahasa Java, Python, C++, C, atau GO yang mengimplementasikan **algoritma *Brute Force*** untuk mencari solusi dalam permainan *Queens*.
- Algoritma *brute force* yang dimaksud harus bersifat murni, tidak menggunakan heuristik apapun.
- **Input:** program akan memberikan pengguna sebuah arahan pada terminal (jika tidak mengerjakan bonus) untuk memilih file test case berekstensi .txt, kemudian program membaca file test case tersebut yang berisi konfigurasi awal dari papan *Queens* yang akan diselesaikan. Perhatikan bahwa input dapat memiliki ukuran papan yang berbeda-beda, serta program harus dapat memvalidasi apakah input yang diberikan merupakan input valid atau bukan.

Berikut adalah contoh file .txt yang akan dijadikan sebagai input:

```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

- **Output:**
  1. Menampilkan hasil akhir dari papan yang sudah terisi oleh *queens* dengan aturan yang benar.
  2. Menampilkan banyak konfigurasi atau iterasi yang ditinjau oleh algoritma.
  3. Menampilkan waktu eksekusi program dalam *millisecond* (cukup waktu pencarian dengan algoritma, tidak termasuk membaca dan menulis file).
  4. *Live Update:* program harus dapat memvisualisasikan proses *brute force* yang dilakukan.

Berikut adalah contoh output berdasarkan contoh input diatas:

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

Waktu pencarian: 120 ms
Banyak kasus yang ditinjau: 1000 kasus
Apakah Anda ingin menyimpan solusi? (Ya/Tidak)
```

2.2 Spesifikasi Bonus

Poin maksimal untuk bonus adalah 10.

- Membuat GUI (*Graphical User Interface* untuk program secara keseluruhan yang dapat memberi *input* dan memberi visualisasi (8 poin).
- *Input as image*, *input* dari program adalah sebuah gambar yang merupakan konfigurasi dari papan awal. (5 poin)
- *Output as image*, *output* dari hasil peletakan *queens* diberikan dalam bentuk gambar. (2 poin)

3 Dasar Teori

3.1 Brute Force

Algoritma *Brute Force* adalah sebuah pendekatan yang menyelesaikan persoalan secara lempang (*straightforward*). Algoritma ini didasarkan langsung pada pernyataan persoalan (*problem statement*) dan definisi konsep yang terlibat. Secara harfiah, kata "*force*" mengindikasikan penggunaan "tenaga" komputasi ketimbang kecerdasan algoritma. Algoritma ini lebih cocok untuk persoalan dengan ukuran masukan yang kecil, karena langkah-langkah penyelesaiannya cenderung sederhana dan implementasinya mudah. Algoritma *brute force* dapat diterapkan untuk memecahkan semua persoalan dan menjamin solusi persoalannya, jika ada.

3.2 Multithreading dalam Java

Multithreading adalah kemampuan sebuah program untuk menjalankan beberapa tugas secara paralel dalam satu waktu. Dalam JavaFX, terdapat satu *thread* utama yang disebut JavaFX Application Thread yang bertanggung jawab untuk menangani UI. *Multithreading* digunakan agar algoritma *brute force* tetap berjalan selagi tampilan di GUI tetap terbarui (*Live Update*). Dalam Java, *multithreading* dapat diimplementasikan dengan membuat objek `Thread` yang sebuah fungsi.

```
    new Thread(() -> {
        // Pemanggilan fungsi untuk Thread utama
        Solver.solve() -> {
            // Kembali ke fungsi lain
        Platform.runLater(() -> Main.GUI());
    });
}).start();
```

4 Desain

4.1 Tech Stack

Aplikasi ini dirancang menggunakan bahasa Java untuk memungkinkan program dapat berjalan di berbagai platform, serta mempermudah perancangan GUI (spesifikasi bonus). Secara garis besar, arsitektur aplikasi terdiri dari komponen-komponen berikut.

1. Bahasa Pemrograman: Java 17
2. Framework GUI: JavaFX
3. Menejemen Dependensi: Apache Maven

4.2 Struktur Program

Proyek disusun mengikuti standar Maven dan spesifikasi tugas:

1. `src/main/`
   (a) `src/main/java/queens/`: Kode sumber Java.
   (b) `src/main/input/`: *Test case* yang digunakan untuk melakukan *testing*.
2. `bin/`: *File executable* atau hasil kompilasi.
3. `doc/`: Spesifikasi dan laporan tugas kecil dalam PDF
4. `pom.xml`: Konfigurasi Maven dan daftar dependensi *library*.
5. `doc/`: Dokumentasi laporan PDF.
6. `pom.xml`: Konfigurasi Maven dan daftar dependensi *library*.

Program utama dibagi menjadi dua komponen utama untuk memisahkan algoritma dengan GUI:

- `queens.Solver` berisi algoritma *brute force* untuk mmenempatkan *queens*. Terdapat logika *solver*
- `queens.Main` berisi komponen UI yang dibangun dengan JavaFX.

4.3 Cara Kompilasi dan Menjalankan Program

Untuk mengompilasi seluruh kode sumber dan mengunduh dependensi JavaFX, jalankan perintah berikut pada direktori *root* projek.

```
mvn clean compile
```

Untuk menjalankan GUI, gunakan JavaFX dengan perintah berikut.

```
mvn javafx:run
```

Untuk menjalankan aplikasi melalui CLI, gunakan perintah berikut.

```
java -cp target/classes queens.Solver
```

5 Implementasi Spesifikasi Wajib

5.1 Algoritma Utama

Algoritma UtamaAlgoritma ini akan mencoba untuk menempatkan satu ratu pada setiap baris secara sistematis dan memeriksa validitasnya. Untuk membantu algoritma ini, dibuat beberapa struktur data, yaitu:

- `Board` digunakan sebagai representasi fisik dari papan permainan. Struktur data ini diimplementasikan menggunakan matriks atau *array* dua dimensi. Setiap elemen dalam *array* menyimpan warna dalam petak.

```
public static String[][] board;
```

- `Koordinat` digunakan untuk menyimpan koordinat pada *board*. `Koordinat` diimplementasikan menggunakan struktur data *record* dari Java. Setiap *record* menyimpan koordinat dalam papan dengan x sebagai baris dan y sebagai kolom.

```
public record Koordinat(int x, int y) {}
```

- `Koordinat[] queenLocation` digunakan untuk menyimpan *set of queens* yang sedang divalidasi.

```
Koordinat[] queenLocation = new Koordinat[n];
for (int i = 0; i < queenLocation.length; i++) {
    queenLocation[i] = new Koordinat(i, 0);
}
```

- *HashSet* `uniqueColor` yang digunakan untuk menyimpan setiap warna yang ada di papan untuk dihitung jumlah warna yang terdapat di dalam papan.

```
Set<String> uniqueColor = new HashSet<>();
```

Berikut adalah langkah-langkah algoritmanya:

1. Program membuat *array of Record* `queenLocation` berukuran N. Di mana setiap elemen array menyimpan koordinat (x, y), di mana x adalah baris $[0..N-1]$ dan y adalah kolom yang awalnya diinisialisasi dengan 0.
2. Program memasuki *infinite loop* dengan `while (true)` untuk mengevaluasi setiap kemungkinan penempatan ratu. Pada setiap iterasi, program mengecek apakah konfigurasi ratu saat ini sudah memenuhi aturan-aturan berikut.
   - Tidak ada dua ratu di kolom yang sama
   - Tidak ada dua ratu di wilayah warna yang sama
   - Tidak ada dua ratu yang saling bersebelahan, termasuk secara diagonal
3. Jika konfigurasi memenuhi aturan, maka algoritma berhenti dan mengembalikan lokasi ratu.
4. Jika konfigurasi tidak memenuhi aturan, maka program akan menggeser ratu
   - Mulai dari baris pertama. Jika kolom belum mencapai indeks maksimal $N-1$, geser kolom ratu ke bawah sebesar 1 petak.

- Jika kolom sudah mencapai indeks maksimal, kembalikan kolom ke 0, lalu pindah ke baris berikutnya.
- Ulangi hingga seluruh kemungkinan teriterasi atau solusi ditemukan.

5. Jika tidak ditemukan solusi hingga akhir, maka program berhenti dan menyatakan tidak ada soluti.

Algoritma ini memiliki kompleksitas waktu algoritma sebesar $O(N^N$, sehingga untuk N yang sangat besar memakan waktu yang sangat lama.

5.2 Algoritma Utama dalam Java

Algoritma Utama dalam Java

```
public static boolean solve(Koordinat[] queenLocation, Consumer<Koordinat[]>
    onStep) {
    int n = board.length;

    while (true) {
        totalKasus++;

        if (validSolution(queenLocation)) {
            return true;
        }

        int baris = 0;
        while (baris < n) {
            int kolom = queenLocation[baris].y;
            if (kolom < n - 1) {
                queenLocation[baris] = new Koordinat(baris, kolom + 1);
                break;
            } else {
                queenLocation[baris] = new Koordinat(baris, 0);
                baris++;
            }
        }

        if (baris == n) {
            break;
        }
    }
    return false;
}
```

5.3 Fungsi Pembantu

Untuk membantu berjalannya program utama, digunakan beberapa fungsi pembantu seperti.

1. `validSolution()`. Fungsi ini mengambil input berupa *list of Record* `Koordinat[] queenLocation` dan mengembalikan `boolean`. Fungsi ini berfungsi untuk mengecek apakah konfigurasi *queens* saat ini sudah memenuhi syarat.

```
    public static boolean validSolution(Koordinat[] queenLocation) { ... }
```

2. `checkColor()`. Fungsi ini mengambil sebuah *record* dan mengembalikan `string` yang berisi label warna dari sebuah koordinat.

```
    public static String checkColor(Koordinat queen) { ... }
```

3. `colorCount()`. Fungsi ini mengembalikan `integer` jumlah total warna yang ada dalam papan.

```
    public static int colorCount() {
        Set<String> uniqueColor = new HashSet<>();
        ...
        return uniqueColor.size();
    }
```

4. `loadBoard()`. Fungsi ini mengambil sebuah `string` berisi *path file* `.txt` dan mengembalikan `boolean` apakah *file* tersebut merupakan konfigurasi papan yang valid.

```
    public static boolean loadBoard(String filePath) {
        ...
        return board.length == colorCount();
    }
```

5. `printBoard()`. Fungsi ini mengambil sebuah konfigurasi ratu dan mencetak papan ke terminal.

```
    public static void printBoard(Koordinat[] queenLocation) {
       if (isQueen) System.out.print("#");
        else System.out.print(board[i][j]);
    }
```

6 Implementasi Spesifikasi Bonus

6.1 GUI

Dalam `Main.java`, struktur komponen digunakan sebagai berikut.

- Stage & Scene sebagai kontainer utama.
- VBox (Root). Layout utama untuk elemen disusun secara vertikal.
- Hbox. Baris horizontal berisi `TextField` untuk *input* nama *file* dan `Button` untuk memulai penyelesaian.
- GridPane. Komponen untuk me-*render* papan.

Untuk memastikan GUI tetap responsif saat menjalankan algoritma, program ini menerapkan *multi-threading* untuk memisahkan perhitungan algoritma dengan menampilkan hasil perhitungan.

1. Algoritma `Solver.solve` dijalankan di dalam *thread* terpisah.
2. `Solver` menerima objek `Consumer`.
3. `Platform.runLater()` memperbarui tampilan papan dari *thread* dari nomor 1.

Fungsi `renderBoard` merupakan fungsi utama yang menampilkan `String[][] board` dari `Solver` dalam GUI.

```
    private void renderBoard(String[][] board, Solver.Koordinat[] solution) {
        boardGrid.getChildren().clear();
        int n = board.length;

        double cellSize = Math.min(600.0 / n, 50.0);
        double fontSize = cellSize * 0.5;

        generateColorMap(board);

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                StackPane cell = new StackPane();
                Rectangle rect = new Rectangle(cellSize, cellSize);
                rect.setFill(colorMap.getOrDefault(board[i][j], Color.LIGHTGRAY
                    ));
                rect.setStroke(Color.BLACK);
                rect.setStrokeWidth(0.2);
                cell.getChildren().add(rect);

                for (Solver.Koordinat q : solution) {
                    if (q.x() == i && q.y() == j) {
                        Label queenLabel = new Label("#");
                        queenLabel.setStyle(
                                "-fx-font-size: " + fontSize + "px; -fx-text-
                                    fill: black;");
```

```
                        cell.getChildren().add(queenLabel);
                    }
                }
                boardGrid.add(cell, j, i);
            }
        }
    }
```

Algoritma ini merubah koordinat dalam `board` dengan *cell* dengan warna yang berbeda-beda yang diberikan oleh fungsi `generateColorMap`.

6.2 Output as Image

```
    private void saveAsImage(String fileName) {
    try {
        WritableImage snapshot = boardGrid.snapshot(null, null);
        File outputFile = new File("src/main/output/" + fileName + ".png");
        ImageIO.write(SwingFXUtils.fromFXImage(snapshot, null), "png",
            outputFile);
        System.out.println("Papan berhasil disimpan sebagai gambar: " +
            outputFile.getAbsolutePath());
    } catch (Exception e) {
        System.out.println("Gagal menyimpan gambar: " + e.getMessage());
    }
}
```

Algoritma ini mengambil tangkapan dari `boardGrid`, lalu menyimpannya dalam *folder* `src/main/output/`. Untuk keseragaman *output* dikonversi menuju format `.png`.

7 Pengujian

Untuk memastikan program berjalan dengan benar, dilakukan beberapa pengujian dengan beberapa kasus uji (*test case*. Kasus uji yang digunakan adalah sebagai berikut.

1. Kasus Uji 4×4
2. Kasus Uji 9×9
3. Kasus Uji 10×10
4. Kasus Uji Tidak Valid
    - Papan Sangat Sempit
    - Jumlah Baris $\neq$ Kolom
    - Jumlah Baris $\neq$ Jumlah Warna
    - *File input* tidak valid

7.1 Kasus Uji 1

```
AABB
AABB
CCDD
CCDD
```

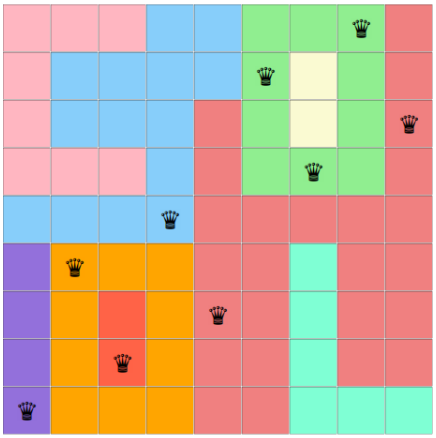
- Deskripsi: Papan 4×4
- Output:

Gambar 3: Hasil Kasus Uji 1

7.2 Kasus Uji 2

```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

- Deskripsi: Papan 9×9
- Output:

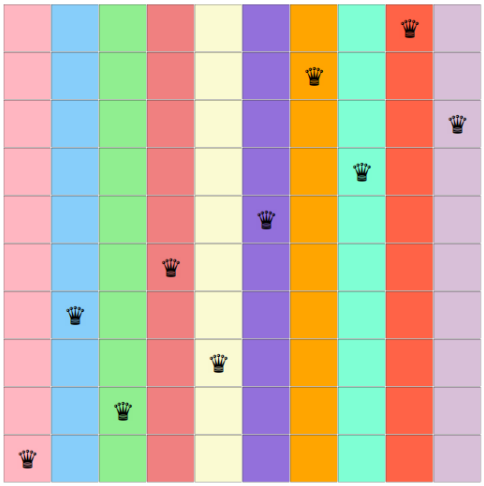


Gambar 4: Hasil Kasus Uji 2

7.3 Kasus Uji 3

```
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
```

- Deskripsi: Papan 10×10
- Output:

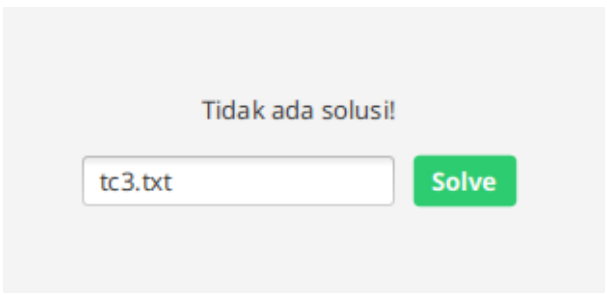


Gambar 5: Hasil Kasus Uji 3

7.4 Kasus Uji 4

```
ABC
ABC
ABC
```

- Deskripsi: Papan 3×3 (Papan terlalu sempit tidak ada solusi)
- Output:

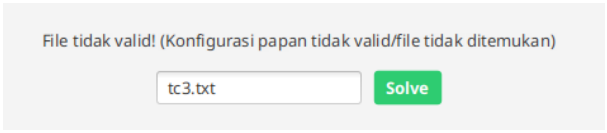


Gambar 6: Hasil Kasus Uji 4

7.5 Kasus Uji 5

```
AA
BB
BB
```

- Deskripsi: Papan 3×2 (Jumlah Baris dan Kolom tidak sama)
- Output:

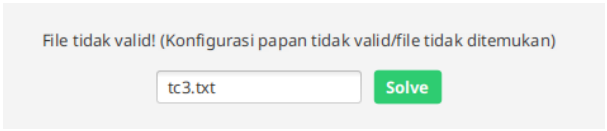


Gambar 7: Hasil Kasus Uji 5

7.6 Kasus Uji 6

```
ABCDD
ABCDD
ABCDD
ABCDD
ABCDD
```

- Deskripsi: Jumlah Baris (5) dengan Jumlah Warna (4) tidak sama
- Output:

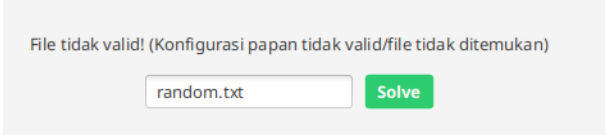


Gambar 8: Hasil Kasus Uji 7

7.7 Kasus Uji 7

```
random.txt
```

- Deskripsi: *File input* tidak valid)
- Output:



Gambar 9: Hasil Kasus Uji 7

7.8 Hasil Pengujian

Berikut adalah ringkasan hasil pengujian kasus uji yang sudah dilakukan beserta waktu penyelesaian dan banyak kasus yang ditinjau.

No	Deskripsi Kasus	Ukuran	Waktu	Banyak Kasus
1	Kasus Normal	4×4	~ 0 ms	115 kasus
2	Kasus Normal	9×9	304 ms	11.775.509 kasus
3	Kasus Normal	10×10	5159 ms	241.357.969 kasus
4	Papan Terlalu Sempit	3×3	N/A	Tidak ada solusi
5	Baris $\neq$ Kolom	3×2	N/A	Input tidak valid
6	Baris $\neq$ Warna	5×5	N/A	Input tidak valid
7	File Tidak Ditemukan	-	N/A	Input tidak valid

Tabel 1: Ringkasan Hasil Pengujian Kasus Uji

8 Penutup

8.1 Lampiran

Tautan kode sumber yang saya rancang dalam program ini ada pada tautan berikut.
Link Github: https://github.com/raulinns/Tucil1-13524044.git

Rangkuman hasil program ini tertera pada tabel berikut.

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	✓	

Tabel 2: Ringkasan Keterselesaian Program

8.2 Kata Penutup

"Tugas Kecil saja sudah seperti ini, bagaimana jika Tugas Besar, ya." Itu yang aku bayangkan saat mengerjakan tugas ini. Namun, Tugas Kecil ini sangat membuka pikiranku tentang algoritma. Selama liburan, aku berpikir bagaimana cara mengurangi penggunaan AI. Melihat mata kuliah Strategi Algoritma ini, tampaknya mata kuliah ini akan menjadi gerbang menuju hal tersebut. Terima kasih, IF2211 Strategi Algoritma!

8.3 Pernyataan

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

Bandung, 18 Februari 2026

Narendra Dharma Wistara M.
13524044