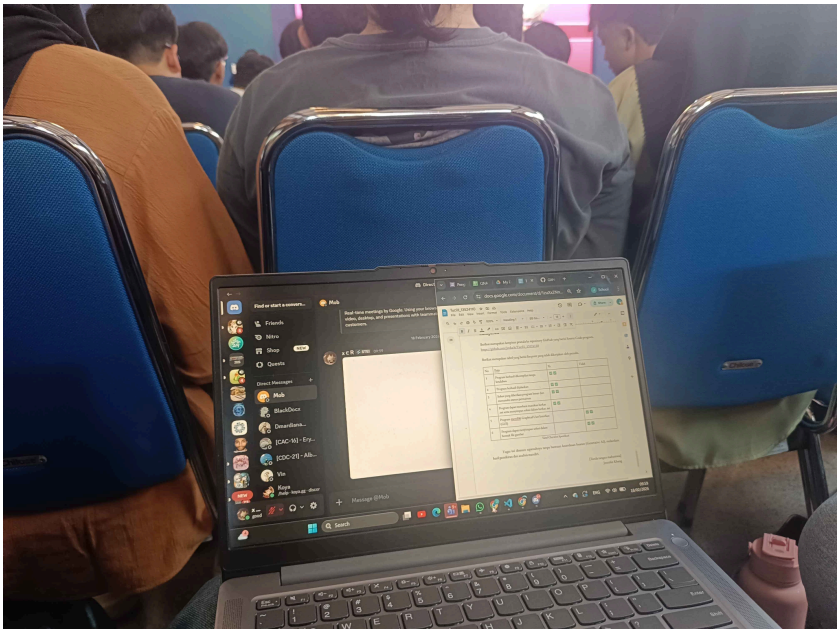
Laporan Tugas Kecil 1
IF2211 Strategi dan Algoritma
Brute Force Algorithm: Penyelesaian Queens LinkedIn

Semester 2 Tahun 2025/2026



"Eh ini ada laporan ya?"

Disusun Oleh:

Jennifer Khang NIM. 13524110

Laboratorium Ilmu dan Rekayasa Komputasi

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

2026

Daftar Isi

Bab I: Pendahuluan

Permainan Queens adalah gim logika yang tersedia pada situs jejaring profesional LinkedIn, sebuah aplikasi yang berhubungan dengan pekerjaan. 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 (8 petak sekitar suatu queen tidak boleh ada bidak lainnya). Penulis membuat program yang dapat menemukan satu solusi penempatan queen pada suatu papan berwarna yang diberikan, atau menampilkan bahwa tidak ada solusi yang valid. Program melakukan pencarian solusi menggunakan algoritma brute force. Pada penugasan ini, penulis menggunakan bahasa Java. Selain itu, penulis hanya mengimplementasikan spesifikasi dasar yang dibutuhkan program, yakni pencarian solusi berbasis .txt dan juga input manual.

Bab II: Penjelasan Implementasi Algoritma Brute Force

Permainan “Queens” bisa diselesaikan dengan banyak sekali cara. Pada tugas kali ini, terdapat *constraints* yang diberikan berupa penggunaan algoritma *pure bruteforce*. Algoritma ini dapat diterapkan dengan cara mengiterasi segala kemungkinan kombinasi posisi bidak ratu pada papan berukuran *N x N*. Pendekatan dilakukan dengan menempatkan bidak queen satu per satu pada tiap baris lalu melakukan validasi di akhir. Namun, ada pendekatan lain yang dapat dilakukan oleh para *problem solver*.

Perhatikan kembali peraturan dari “Queens”. Berikut adalah rincian dari tiap aturan yang dimiliki oleh permainan ini.

1. Setiap ratu yang berada pada baris atau kolom yang berbeda.
2. Setiap ratu tidak boleh bersentuhan secara diagonal maupun ortogonal (8 kotak disekitarnya).
3. Setiap wilayah warna yang didefinisikan pada papan hanya boleh memiliki tepat satu ratu.

Ada beberapa peraturan yang tak tertulis pada 3 peraturan tersebut, yakni sebagai berikut.

1. Setiap baris mesti memiliki tepat satu ratu, maka jumlah ratu akan sama dengan ukuran N dari papan bidak.
2. Setiap ratu harus berada pada *region* atau daerah warna yang berbeda-beda, maka daerah warna juga harus berjumlah ukuran N seperti pada papan bidak.

Cara lain yang digunakan oleh penulis adalah dengan berfokus kepada warna daripada baris. Tentu, ini bukanlah *heuristic* karena merupakan peraturan alami dari permainan “Queens” dan tidak berdampak pada jumlah kompleksitas O(n) dari suatu *test case*. Berikut adalah langkah-langkah detail dari algoritma *brute force* yang diimplementasikan pada fungsi *solve* dan *isSolved* pada kelas *method*.

1. Program menerima input berupa *grid* berukuran *N x N*.
2. Algoritma pertama kali menyusun sebuah list *colorOrder* berisi warna(karakter) beserta koordinat, mulai dari warna yang paling awal muncul.
3. Algoritma kemudian menyusun bidak ratu berdasarkan *region* atau daerah yang muncul duluan berdasarkan aturan baris (*row-based*). Setelah selesai menaruh sejumlah *N* bidak ratu, program melakukan validasi pada papan apakah konfigurasi memenuhi syarat-syarat yang berlaku, yaitu tepat satu ratu pada setiap baris dan kolom (tidak perlu memvalidasi warna), sehingga memakai *exhaustive search*. Berikut merupakan cara validasi papan bidak.
    - Posisi bidak ratu disimpan dalam List of item bernama placedQueens.

- Program akan melakukan iterasi n^2 pada fungsi *isSolved*, mencari bidak lainnya secara iterasi dengan model objek *board.cell[][].*
- Program juga akan memeriksa apakah ratu bersentuhan secara horizontal, vertikal, maupun diagonal sejauh petak.

4. Apabila telah ditemukan solusi yang memenuhi syarat, program akan menampilkan solusi dari algoritma dan menampilkan waktu serta jumlah iterasi. Jika tidak valid, program akan mencetak pesan validasi.

Dengan melihat implementasi algoritma pada fungsi *isSolved* dan *solve*, algoritma akan melakukan iterasi dengan maksimum sebanyak. Maka dari itu, program memiliki kompleksitas waktu sebesar O(n^n). Brute force hanya akan cocok digunakan apabila papan bidak berukuran N kecil.

Fungsi validasi yang digunakan dalam mengecek syarat-syarat dari papan yang valid adalah sebagai berikut.

Algoritma 1 Algoritma Validasi Papan Bidak

```
FUNCTION IsSolved(placedQueens, totalColors):

  IF size(placedQueens) ≠ totalColors THEN
    RETURN false

  FOR i FROM 0 TO size(placedQueens)-1:

    row ← placedQueens[i].row
    col ← placedQueens[i].col

    FOR j FROM 0 TO size(placedQueens)-1:

      IF i == j THEN
        CONTINUE

      jRow ← placedQueens[j].row
      jCol ← placedQueens[j].col

      IF jRow == row THEN
        RETURN false

      IF jCol == col THEN
        RETURN false
```

```
      IF |jRow - row| ≤ 1 AND |jCol - col| ≤ 1 THEN
        RETURN false

  RETURN true
```

Berikut merupakan penjelasan algoritma brute force yang direpresentasikan dalam pseudocode.

Algoritma 2 Algoritma Brute Force Queens LinkedIn

```
FUNCTION Solve():

  colorOrder ← BuildColorOrder()
  totalColors ← size(colorOrder)

  CREATE array pointer[totalColors]
  SET all pointer values to 0

  iteration ← 0

  WHILE true:

    placedQueens ← empty list

    FOR c FROM 0 TO totalColors-1:
      q ← colorOrder[c][ pointer[c] ]
      ADD q to placedQueens

    iteration ← iteration + 1

    IF IsSolved(placedQueens, totalColors) THEN
      PRINT solution
      RETURN true

    carry ← totalColors - 1

    WHILE carry ≥ 0:

      pointer[carry] ← pointer[carry] + 1

      IF pointer[carry] < size(colorOrder[carry]) THEN
        BREAK

      pointer[carry] ← 0
```

```
    carry ← carry - 1

  IF carry < 0 THEN
    PRINT "No solution"
    RETURN false
```

Bab 3: Source Code Program

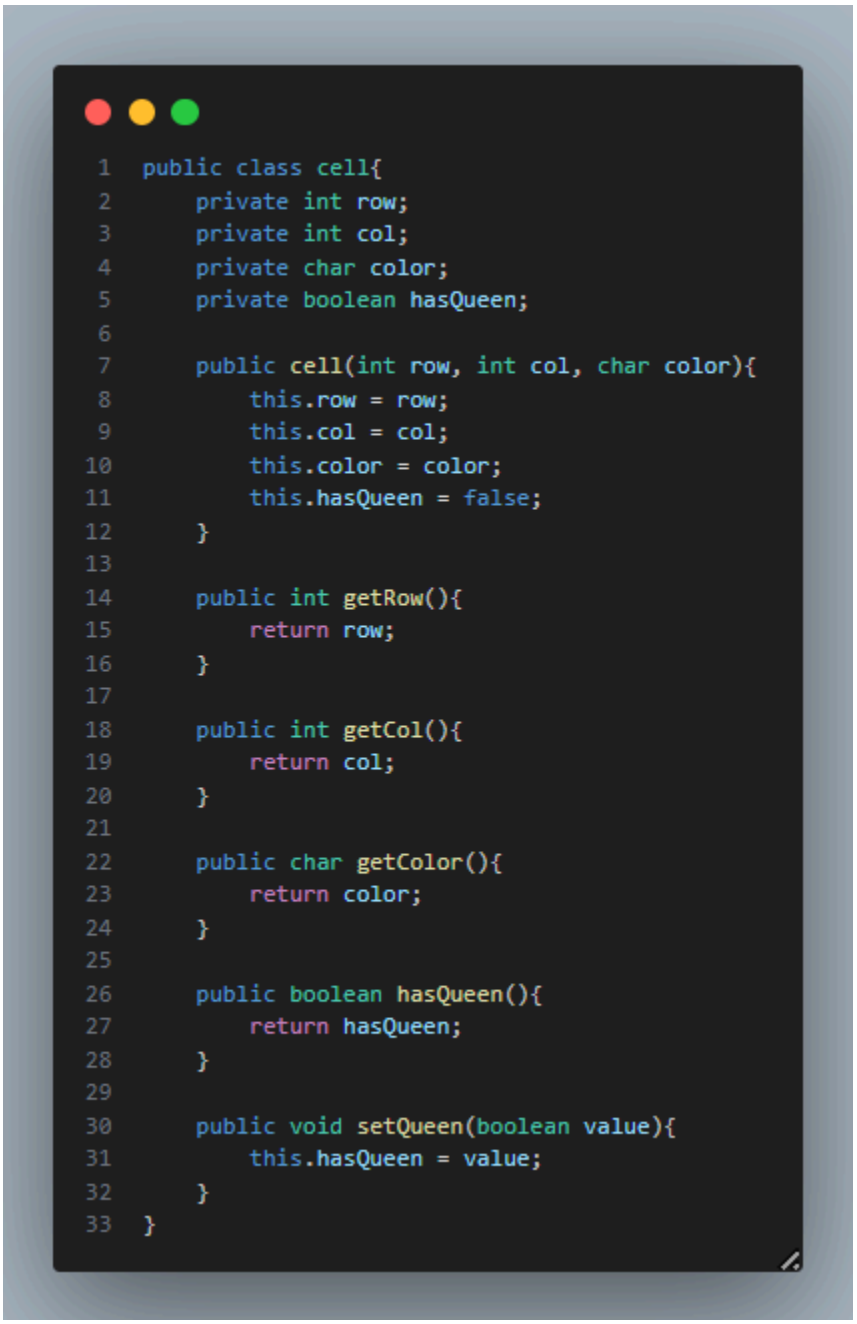
Source Code program yang dibuat penulis memiliki struktur folder sebagai berikut.

```
Tucil1_13524110/
│   README.md
│
├── bin/
│   ├── boardMatrix.class
│   ├── cell.class
│   ├── item.class
│   ├── main.class
│   ├── method.class
│   └── outPath/
│
├── doc/
│   └── Tucil1_13524110.pdf
│
├── src/
│   └── java/
│       ├── boardMatrix.java
│       ├── cell.java
│       ├── item.java
│       ├── main.java
│       └── method.java
│
├── test/
│   ├── 1.txt
│   ├── 2.txt
│   ├── 3.txt
│   ├── 4.txt
│   └── 5.txt
│
```

Berikutnya, akan dijelaskan isi serta penjelasan dari tiap-tiap file.

1. cell.java

Berikut merupakan isi dari kelas *cell* yang akan digunakan sebagai struktur dasar dalam menyimpan informasi terkait papan catur per kotak.



```java
public class cell{
    private int row;
    private int col;
    private char color;
    private boolean hasQueen;

    public cell(int row, int col, char color){
        this.row = row;
        this.col = col;
        this.color = color;
        this.hasQueen = false;
    }

    public int getRow(){
        return row;
    }

    public int getCol(){
        return col;
    }

    public char getColor(){
        return color;
    }

    public boolean hasQueen(){
        return hasQueen;
    }

    public void setQueen(boolean value){
        this.hasQueen = value;
    }
}
```

Selain itu, terdapat fungsi seperti selektor dan konstruktor yang dapat dilihat pada *source code*.

2. item.java

Berikut merupakan isi dari kelas *item* yang akan menyimpan koordinat dari suatu objek. Kelas ini akan digunakan untuk struktur data dasar dari List/Array dari warna serta pemetaan bidak ratu.

```java
public class item {
    private int row;
    private int col;

    public item(int row, int col) {
        this.row = row;
        this.col = col;
    }

    public int getRow(){return row;}
    public int getCol(){return col;}
}
```

3. boardMatrix.java

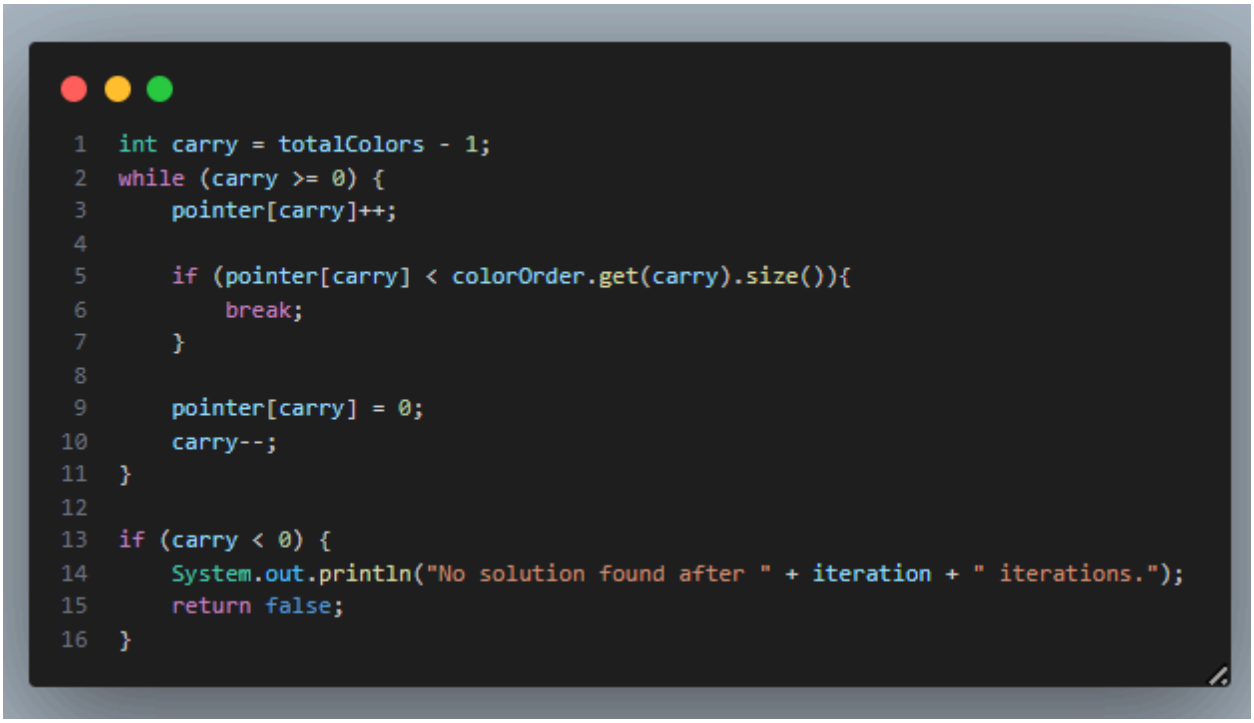
Kelas *boardMatrix* berisi struktur penyimpanan objek dengan kelas *cell* yang akan mengkonstruksi dasar dari suatu papan bidak. Terdapat beberapa fungsi dasar, seperti memvalidasi papan bidak, penulisan papan bidak, serta konversi string menjadi struktur data matriks. Matriks ini sendiri juga dapat diartikan sebagai *array* dengan dua dimensi.

```java
public static String[] splitRow(String line) {
    line = line.trim();
    line = line.replace(" ", "");
    return line.split("");
}
```

Gambar sebelumnya merupakan salah satu fungsi unik pada file ini, di mana input User akan secara otomatis diproses per karakter meski memiliki spasi yang panjang.

4. method.java

File ini berisi metode yang akan digunakan dalam penyelesaian permasalahan. Fungsi-fungsi dari kelas ini sudah dibahas pada bab sebelumnya namun akan dijelaskan lebih lanjut pada bab ini. Kelas *method* memiliki atribut board (representasi papan) beserta nInterval (setiap angka ke-N akan diprint). Salah satu hal yang unik dari kode dalam file ini adalah sebagai berikut.



```
int carry = totalColors - 1;
while (carry >= 0) {
    pointer[carry]++;

    if (pointer[carry] < colorOrder.get(carry).size()){
        break;
    }

    pointer[carry] = 0;
    carry--;
}

if (carry < 0) {
    System.out.println("No solution found after " + iteration + " iterations.");
    return false;
}
```

Sistem carry ini memungkinkan program untuk mencoba semua kombinasi yang memungkinkan secara sistematis.

5. main.java

Pada file ini, *main* merupakan program dasar yang menjadi kerangka dari program ini. Alurnya adalah sebagai berikut.

   a. Meminta input board, baik dari file .txt atau input manual user

```java
Scanner sc = new Scanner(System.in);
System.out.println("Input mode:");
System.out.println("1. Manual input");
System.out.println("2. Read from .txt file");
System.out.print("Choose: ");
int inputMode = Integer.parseInt(sc.nextLine().trim());
```

b. Validasi board input

c. Menjalankan solver dari kelas method serta mengukur waktu eksekusi dari library bawaan *java*

```java
method solver = new method(board, interval);
long startTime = System.currentTimeMillis();
boolean solved = solver.solve(interval);
long endTime = System.currentTimeMillis();
```

d. Menyimpan hasil dalam file .txt (opsional)

```java
if (solved) {
    System.out.print("Save result to .txt file? (y/n): ");
    String saveLine = sc.nextLine().trim();
    while (saveLine.isEmpty()){
        saveLine = sc.nextLine().trim();
    }
    char save = saveLine.charAt(0);

    if (save == 'y' || save == 'Y') {
        System.out.print("Enter output .txt file path: ");
        String outPath = sc.nextLine().trim();

        while(!(isValidOutputPath(outPath))){
            System.out.print("\nEnter again output .txt file path: ");
            outPath = sc.nextLine().trim();
        }

        BufferedWriter writer = new BufferedWriter(new FileWriter(outPath));
        writer.write(board.tulisBoardMatrix(board));
        writer.close();
        System.out.println("Saved!\n");
    }
}
```

e. Mengulang jika user mau

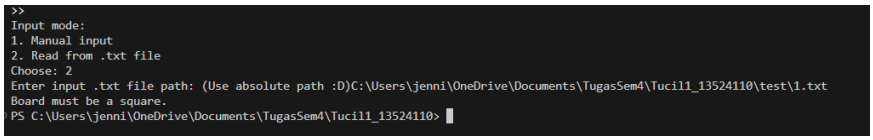
```java
System.out.println("Retry? (y/n)");
char x = sc.next().charAt(0);

if (x == 'n' || x == 'N') break;
```

Bab 4: Eksperimen

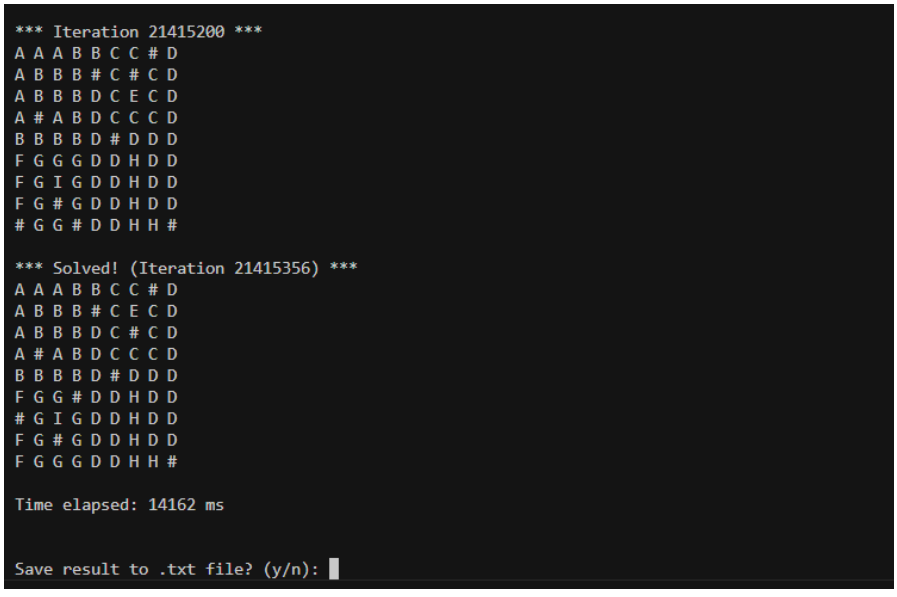
Bab ini merupakan tangkapan layar yang memperlihatkan input dan output beserta penjelasannya.

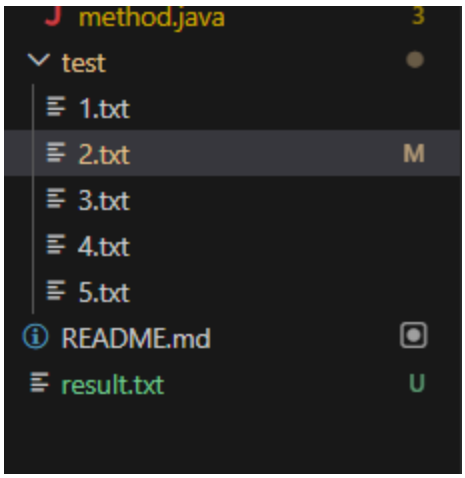
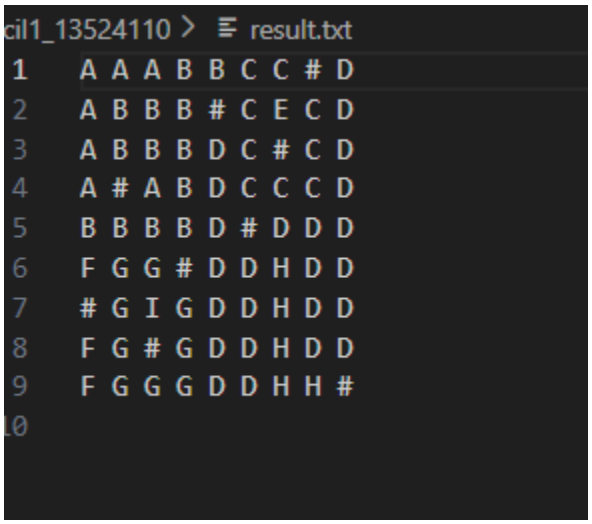
1. Test Case 1

Input	Output
ABCCCCCC BBBDEECC FBDDEECC FDDEEEEC FDEEEEEC FEEFEE GC	

Hal ini benar karena input dari User tidak berupa papan bidak yang berbentuk persegi atau $N \, x \, N$. Program masih menerima input hanya saja mencetak validasi terkait ukuran dari papan bidak yang dimasukkan.

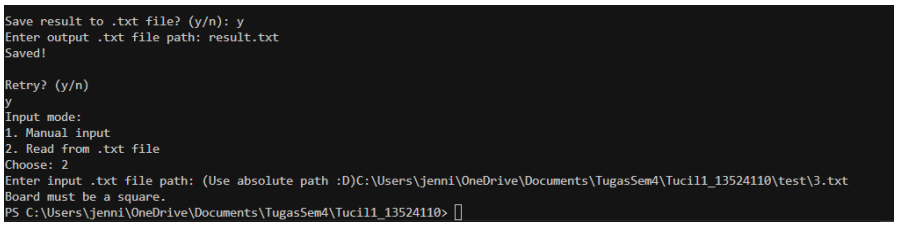
2. Test Case 2

Input	Output
AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH	

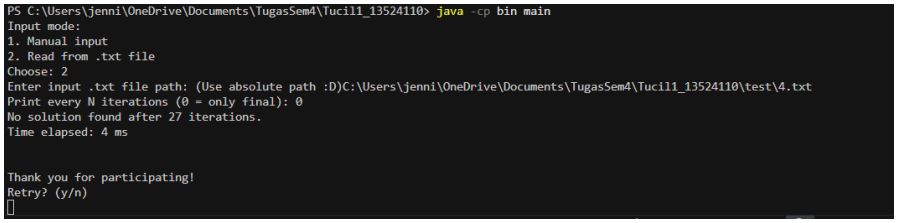
Program berhasil mendapatkan solusi dan benar. Program juga dapat menyimpan ke dalam bentuk .txt.

3. Test Case 3

Input	Output
AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD	

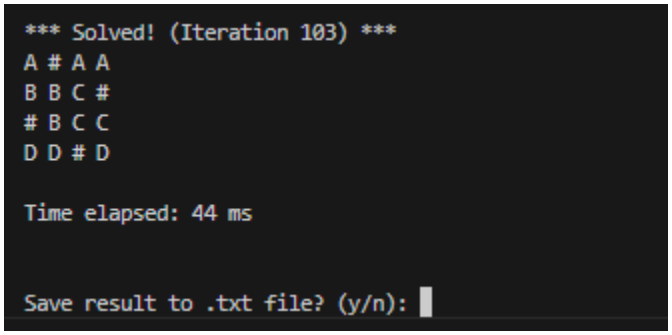
User memasukkan papan bidak berukuran *9 x 6* sehingga program tidak berjalan dan mencetak pesan validasi.

4. Test Case 4

Input	Output
AAA BBC BCC	

Terlihat bahwa tidak ditemukan solusi karena, secara logika, bidak ratu akan terus bertetanggaan dengan ratu lainnya dalam kasus bidak papan *3 x 3*.

5. Test Case 5

Input	Output
AAAA BBCC BBCC DDDD	

Program berhasil dijalankan.

Bab 5: Kesimpulan

Algoritma *brute force* merupakan salah satu algoritma *naif* yang dapat diimplementasikan. Algoritma ini sendiri tidak melakukan pemikiran awal dan langsung diimplementasikan tanpa pemikiran panjang. Tentu, pemikiran *naif* ini berakibat pada tingginya biaya komputasi sehingga program penyelesaian problematika "Queens" dapat mencapai kompleksitas $O(n^n)$ seiring bertambahnya ukuran papan bidak. Jadi, dapat disimpulkan bahwa cara ini merupakan cara paling simpel namun mahal akibat dari kompleksitas waktu yang dapat meningkat secara eksponensial.

Lampiran

Berikut merupakan lampiran pranala ke repository GitHub yang berisi Source Code program.
https://github.com/jenka-h/Tucil1_13524110

Berikut merupakan tabel yang berisi list poin yang telah dikerjakan oleh penulis.

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✅✅	
2	Program berhasil dijalankan	✅✅	
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 Checklist Spesifikasi

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

Jennifer Khang