

# **Laporan Tugas Kecil 1**

## **IF2211 Strategi Algoritma**

Penyelesaian Permainan *Queens Linkedin*



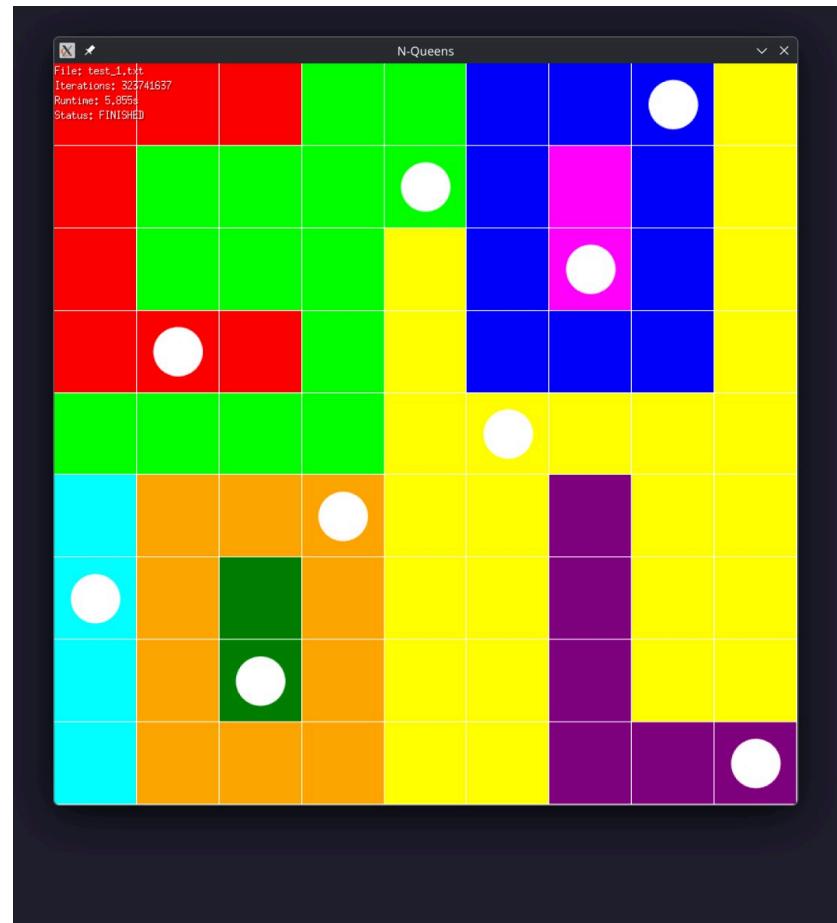
Aufa Rienaldifaza Ahmad  
NIM 13524027

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

<b>1. Permasalahan</b>	<b>3</b>
<b>2. Brute Force</b>	<b>5</b>
<b>3. Penurunan Solusi</b>	<b>5</b>
<b>4. Abstraksi</b>	<b>5</b>
<b>5. Pseudocode</b>	<b>5</b>
<b>6. Implementasi</b>	<b>6</b>
<b>7. Dokumentasi Test-Case</b>	<b>11</b>
6.1. Test-Case 1	11
6.2. Test-Case 2	11
6.3. Test-Case 3	12
6.4. Test-Case 4	13
6.5. Test-Case 5	13
6.6. Test-Case 6	14
<b>LAMPIRAN</b>	<b>15</b>

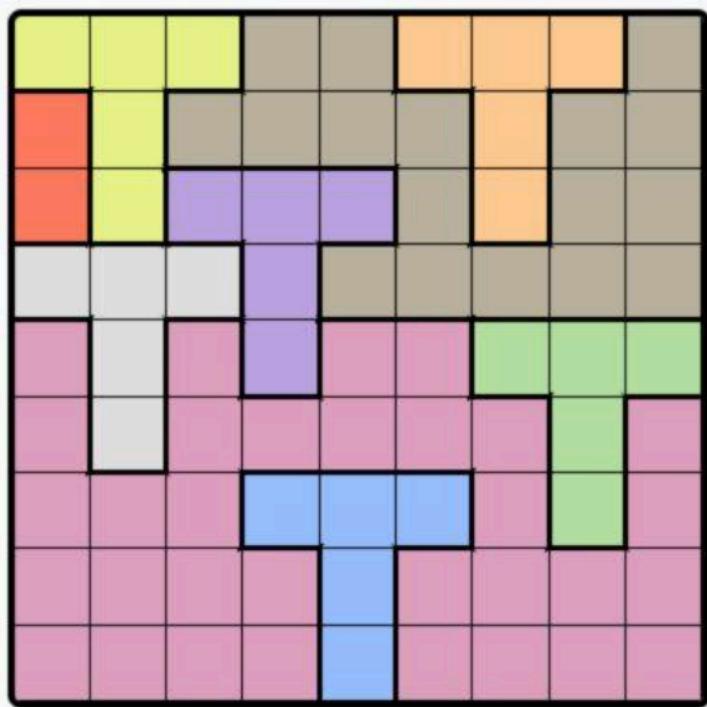
## 1. Permasalahan

*Queens* merupakan gim logika yang bertujuan untuk menempatkan *queen* pada sebuah papan persegi berwarna sehingga terdapat hanya **satu queen** pada tiap **baris, kolom, dan daerah warna, termasuk secara diagonal**.



Gambar 1. Contoh Solusi dari Gim *Queens*

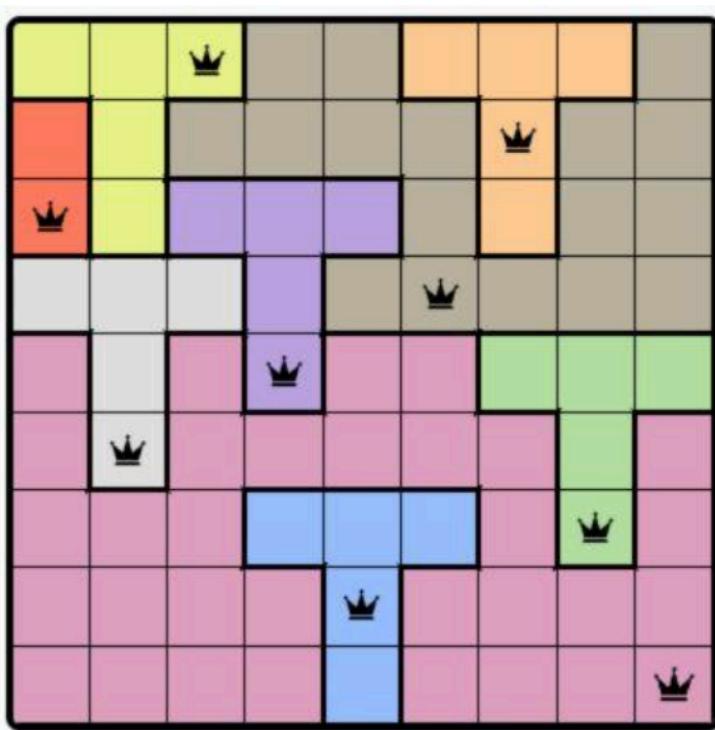
Contohnya, diberikan sebuah papan sebagai berikut.



Gambar 2. Permasalahan Gim *Queens*

Dapat ditemukan sebuah solusi yang memenuhi:

1. Terdapat satu *queen* pada setiap warna,
2. Terdapat satu *queen* pada setiap kolom,
3. Terdapat satu *queen* pada setiap baris,



Gambar 3. Solusi permasalahan *Queens*

Perhatikan bahwa pada suatu papan NxN, **harus terdapat N buah queen**.

## 2. Brute Force

*Brute force* merupakan algoritma yang menyelesaikan persoalan secara lempeng (*straightforward*). Algoritma *brute force* memecahkan persoalan secara sederhana dengan langkah-langkah yang mudah dipahami.

## 3. Penurunan Solusi

Permasalahan Gim *Queens* dapat didekomposisi menjadi dua permasalahan. Pertama, pada suatu *cell* apakah *queen* dapat diletakkan sesuai aturan atau tidak. Kedua, pengecekan apakah masih terdapat konfigurasi posisi (*state*) yang dapat ditelusuri. Oleh karena itu, terdapat dua fungsi yang diperlukan untuk menyelesaikan permasalahan ini.

Pertama, pengecekan posisi N buah *queen* yang valid. Hal ini dilakukan dengan mengecek tiga kondisi:

1. Apakah terdapat *queen* di baris dan kolom pada satu *cell*,
2. Apakah terdapat *queen* di warna yang sama,
3. Apakah terdapat *queen* tetangga, yaitu 8 *cell* yang mengitari suatu *queen*.

Kedua, konfigurasi posisi *queen* yang dapat dicoba. Hal ini dilakukan dengan mengubah posisi *queen* pada suatu baris untuk “maju” ke kolom selanjutnya sehingga suatu *queen* akan mencoba semua kombinasi kolom pada suatu baris. Apabila suatu *queen* bisa terletak pada seluruh kolom pada suatu baris, *queen* tersebut akan kembali ke posisi awal.

Kedua fungsi ini digabung dalam sebuah perulangan kondisional yang akan membuat setiap *queen* mencoba seluruh konfigurasi sampai **CheckValid** mengembalikan *true* untuk satu *state* N buah *queen* yang diberikan oleh **NextState**.

## 4. Abstraksi

Berdasarkan peraturan permainan yang ada, dapat dibentuk sebuah struktur data yang merepresentasikan posisi *queen*. Diketahui terdapat N buah *queen* untuk N buah baris dan kolom. Dari sini, dapat dibentuk sebuah struktur data *array* yang merepresentasikan posisi *queen* pada baris ke-*i* sehingga kita hanya perlu memanipulasi posisi kolom *queen* pada baris ke-*i* dan tidak perlu melakukan *sequential search* secara berulang. Berikut struktur data *array* dari posisi *queen* yang terdapat pada **struct Board**.

```
Q []int
```

## 5. Pseudocode

```
function CheckValid(board) -> boolean
    n <- board.Size
    q <- board.Q
    grid <- board.Grid

    i traversal [0..n-1]
        j traversal [i+1..n-1]
            IF q[i] = q[j] then return false

            IF grid[i][q[i]] = grid[j][q[j]] then return
```

```

false

    deltaRow <- abs(i - j)
    deltaCol <- abs(q[i] - q[j])

    IF deltaRow = 1 AND deltaCol <= 1 then
return false

```

```

function NextState(board) -> boolean
    n <- board.Size
    q <- board.Q

    i <- n - 1

    REPEAT
        IF q[i] < n - 1 then
            q[i] <- q[i] + 1
            return true

        ELSE
            q[i] <- 0
            i <- i - 1

    UNTIL (i < 0)
return false

```

## 6. Implementasi

Implementasi dilakukan secara modular dengan membuat *package-package* berdasarkan fungsinya. Terdapat empat *package* yang diimplementasikan, yaitu

### 1. Models

*Package* ini berfungsi untuk membuat **struct Board** yang akan diinisiasi dan dimanipulasi selama berjalannya program.

```

type Board struct {
    Size      int
    Grid      [][]string
    Q         []int
    Iter      int
    Solutions [][]int
    SolCount  int
}

```

Di dalam *package* ini, terdapat juga sebuah metode **NewBoard** yang digunakan untuk meniginisiasi **struct Board**.

```

func NewBoard(size int, gridData [][]string) *Board {
    return &Board{
        Size:      size,
        Grid:      gridData,
        Q:         make([]int, size), // jumlah Q = n (n
x n grid)
        Solutions: make([][]int, 0),
    }
}

```

## 2. Utils

*Package* ini berfungsi untuk melakukan pembacaan dan penulisan *file* (.txt). Terdapat dua fungsi dalam *package* ini, yaitu **ReadFile** dan **WriteFile**.

```

func ReadFile(path string) ([][]string, error) {
    file, err := os.Open(path)
    if err != nil {
        log.Printf("couldn't open file %s\n", path)
    }
    defer file.Close()
    var data [][]string
    scanner := bufio.NewScanner(file)
    for scanner.Scan() {
        line := scanner.Text()
        line = strings.TrimSpace(line)
        rowChars := strings.Split(line, " ")
        if len(rowChars) > 0 {
            data = append(data, rowChars)
        }
    }
    if err := scanner.Err(); err != nil {
        return nil, err
    }
    return data, nil
}
func WriteFile(filename string, board *models.Board) error {
    file, err := os.Create(filename)
    if err != nil {
        return err
    }
    defer file.Close()
}

```

```

size := board.Size
queens := board.Solutions[0]

for r := 0; r < size; r++ {
    line := ""
    for c := 0; c < size; c++ {
        if queens[r] == c {
            line += "# "
        } else {
            line += board.Grid[r][c] + " "
        }
    }
    fmt.Fprintln(file, line)
}
return nil
}

```

### 3. Solver

*Package* ini berisi metode yang digunakan untuk memanipulasi **struct Board** menggunakan algoritma *brute-force*. **CheckValid** merupakan implementasi dari penurunan solusi bagian 1, **NextState** merupakan implementasi dari penurunan solusi bagian 2.

```

func CheckValid(board *models.Board) bool {
    n := board.Size
    q := board.Q
    grid := board.Grid

    for i := 0; i < n; i++ {
        for j := i + 1; j < n; j++ {
            // if columns are the same -> false
            if q[i] == q[j] {
                return false
            }

            // if same color -> false
            if grid[i][q[i]] == grid[j][q[j]] {
                return false
            }
        }
    }
}

```

```

        }

        //check adjacency
        if math.Abs(float64(i-j)) == 1 &&
math.Abs(float64(q[i]-q[j])) <= 1 {
            return false
        }
    }
    return true
}

func NextState(board *models.Board) bool {
    n := board.Size
    q := board.Q

    for i := n - 1; i >= 0; i-- {
        if q[i] < n-1 {
            q[i]++ // check for each column in a row
            return true
        } else {
            q[i] = 0
        }
    }
    return false
}

```

#### 4. GUI

*Package* GUI berfungsi untuk menampilkan struktur *board*, *state* dari *board* setiap iterasi ke-100.000, hasil akhir dari *board*, waktu berjalannya program, dan jumlah iterasi yang berjalan. *Package* ini dibuat dengan *library* Golang bernama Ebiten.

Terdapat sebuah **struct Game** yang merupakan *entity* untuk merepresentasikan **board** dalam bentuk gambar.

```

type Game struct {
    Board           *models.Board
    Found          bool
    IterationsPerFrame int
    TileSize       float32
    InputName      string
    StartTime      time.Time
}

```

```

    EndTime           time.Duration
}

```

Pada *package* ini, juga dilakukan pengubahan bacaan metode **ReadFile** pada *package* utils untuk menjadi sejumlah warna yang di-*hardcode* menjadi sebuah *map*.

```

var palette = map[rune]color.RGBA{
    'A': {255, 0, 0, 255},
    'B': {0, 255, 0, 255},
    'C': {0, 0, 255, 255},
    'D': {255, 255, 0, 255},
    'E': {255, 0, 255, 255},
    'F': {0, 255, 255, 255},
    'G': {255, 165, 0, 255},
    'H': {128, 0, 128, 255},
    'I': {0, 128, 0, 255},
    'J': {165, 42, 42, 255},
    'K': {255, 192, 203, 255},
    'L': {128, 128, 128, 255},
    'M': {0, 0, 128, 255},
    'N': {128, 128, 0, 255},
    'O': {0, 128, 128, 255},
    'P': {192, 192, 192, 255},
}

```

Berikut merupakan penjelasan metode-metode pengoperasian GUI yang terdapat dalam *package* ini.

### **StartGame (grid [][]string, originalPath string)**

Fungsi ini bertugas memvalidasi input sebelum visualisasi dimulai. Program memastikan papan berukuran  $N \times N$  dan hanya mengandung huruf (menggunakan pengecekan `unicode.IsLetter`). Jika validasi lolos, fungsi ini menginisialisasi window Ebiten dan memulai game loop.

### **Draw (screen \*ebiten.Image)**

Fungsi ini dipanggil setiap frame untuk menggambar keadaan papan pada suatu *state*. Setiap sel pada papan diwarnai berdasarkan karakter inputnya menggunakan *map* warna. Jika sebuah sel ditempati oleh *Queen*, program akan menggambar lingkaran di *cell* tersebut. Informasi *real-time* seperti nama file, jumlah iterasi, dan durasi eksekusi ditampilkan di layar menggunakan `ebitenutil.DebugPrint`.

### **Update()**

Ini adalah inti dari game loop. Setiap iterasi memanggil `solver.NextState` untuk mencari kemungkinan posisi *Queen* berikutnya. Jika `solver.CheckValid`

bernilai true (solusi ditemukan), program menyimpan hasil ke file (.txt) dan gambar (.png) ke dalam folder output, serta menghentikan perhitungan waktu.

### SaveImage (path string)

Fungsi ini memungkinkan program untuk merender ulang keadaan papan terakhir ke dalam memori dan menyimpannya sebagai file PNG.

## 7. Dokumentasi Test-Case

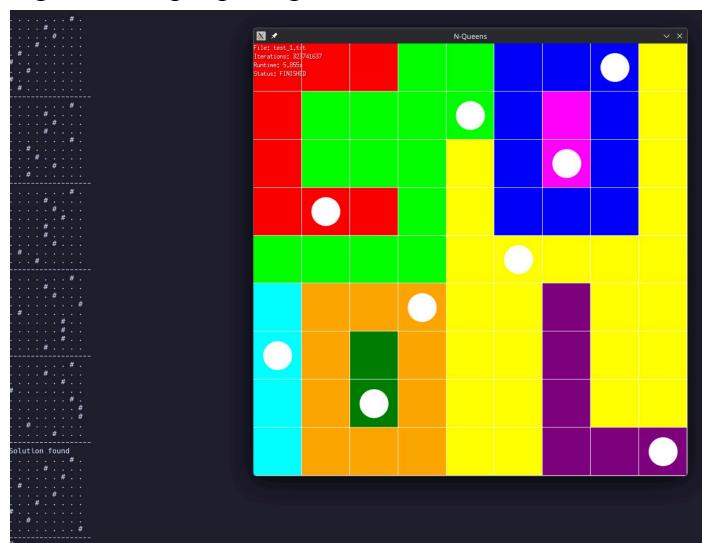
### 6.1. Test-Case 1

Masukan berupa *grid* 9x9 dengan struktur sebagai berikut.

AAABBBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH

Gambar 4. Masukan Test Case 1

Keluaran dari program terdapat pada gambar berikut.



Gambar 5. Keluaran Test Case 1

Program berjalan selama 5,855 detik (termasuk dengan *update state* pada GUI).

### 6.2. Test-Case 2

Masukan berupa *grid* 8x8 dengan struktur sebagai berikut.

```
AABCDDDE  
AABCCDDE  
FAAAADDE  
FFAAAAADE  
GGAAAADA  
HGAAAAAA  
HHHAAAAAA  
HHHHHAAA
```

Gambar 6. Masukan *Test Case 2*

Keluaran dari program terdapat pada gambar berikut.



Gambar 7. Keluaran *Test Case 2*

Program berjalan selama 460 milidetik (termasuk dengan *update state* pada GUI).

### 6.3. *Test-Case 3*

Masukan berupa *grid* 3x3 dengan struktur sebagai berikut.

```
ABC  
ABC  
ABC
```

Gambar 8. Masukan *Test Case 3*

Keluaran dari program terdapat pada gambar berikut.

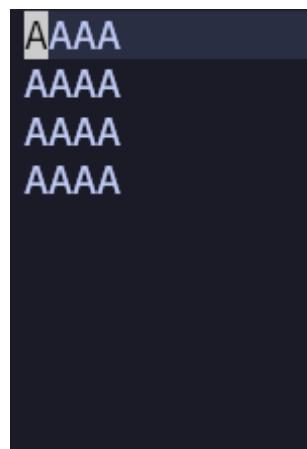
```
[fazabobi@arch tucili-stima]$ make run
go run src/main.go
Enter the input file path (e.g., test/files/test_1.txt): test/files/test_8.txt
solve using pure brute force (with GUI) or with heuristic (more fun)?
1 for brute force (+GUI), 2 for heuristic
1
2026/02/17 22:59:53 unsolvable, no solutions for size %v3
```

Gambar 9. Keluaran Test Case 3

Program tidak berjalan dan memberikan *Fatal Error* dengan pesan bahwa tidak akan ada solusi untuk papan dengan jumlah baris dan kolom sebanyak tiga.

#### 6.4. Test-Case 4

Masukan berupa *grid* 4x4 dengan jumlah warna hanya satu.



Gambar 10. Masukan Test Case 4

Keluaran dari program terdapat pada gambar berikut.

```
make: *** [Makefile:5: run] Error 1
[fazabobi@arch tucili-stima]$ make run
go run src/main.go
Enter the input file path (e.g., test/files/test_1.txt): test/files/test_9.txt
solve using pure brute force (with GUI) or with heuristic (more fun)?
1 for brute force (+GUI), 2 for heuristic

UNSOLVABLE: only 1 colors found, but 4 are required
algorithm ran for 36.211μsfound solution
```

Gambar 11. Keluaran Test Case 4

Program tidak berjalan dan memberikan *Fatal Error* dengan pesan bahwa jumlah warna tidak sama dengan jumlah baris dan kolom.

#### 6.5. Test-Case 5

Masukan berupa file .txt kosong.



Gambar 12. Masukan *Test Case 5*

Keluaran dari program terdapat pada gambar berikut.

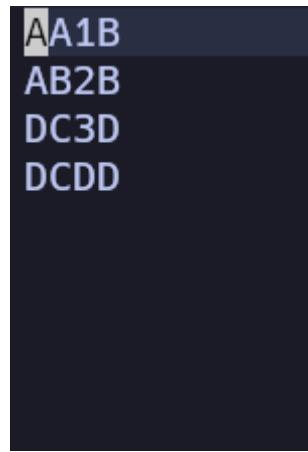
```
Enter the input file path (e.g., test/files/test_1.txt): test/files/test_err_empty.txt
solve using pure brute force (with GUI) or with heuristic (more fun)?
1 for brute force (+GUI), 2 for heuristic
1
2026/02/17 23:03:05 board is empty
exit status 1
```

Gambar 13. Keluaran *Test Case 5*

Program tidak berjalan dan memberikan *Fatal Error* dengan pesan *grid* masukan kosong.

#### 6.6. *Test-Case 6*

Masukan berupa *edge case* berupa angka dalam *grid*.



Gambar 14. Masukan *Test Case 6*.

Keluaran dari program terdapat pada gambar berikut.

```
go run src/main.go
Enter the input file path (e.g., test/files/test_1.txt): test/files/test_err_invalid_char.txt
solve using pure brute force (with GUI) or with heuristic (more fun)?
1 for brute force (+GUI), 2 for heuristic
1
2026/02/17 23:12:32 Error: Invalid character '1' .
exit status 1
```

Gambar 15. Keluaran *Test Case 6*

Program tidak berjalan dan memberikan *Fatal Error* dengan pesan karakter tidak valid.

## LAMPIRAN

*Source code* dari program yang terdapat pada *link* github *repository* berikut  
[https://github.com/aufafaza/Tucil1\\_13524027/tree/main](https://github.com/aufafaza/Tucil1_13524027/tree/main)

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



Aufa Rienaldifaza Ahmad  
NIM13524027