

**IF2211 - Strategi Algoritma  
Tugas Kecil  
Penyelesaian Permainan Queens Linkedin**



Oleh:  
Muhammad Naufal Romi Annafi  
K02 - 13524058

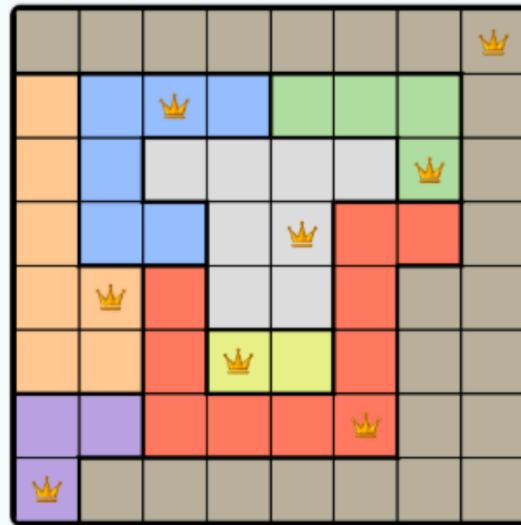
**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
JL. GANESA 10, BANDUNG 40132  
2026**

## Daftar Isi

<b>Daftar Isi.....</b>	<b>2</b>
<b>1. Pendahuluan.....</b>	<b>3</b>
1.1. Deskripsi Tugas.....	3
1.2. Ilustrasi Kasus.....	4
<b>2. Pembahasan.....</b>	<b>5</b>
<b>3. Implementasi.....</b>	<b>7</b>
3.1. Bahasa Pemrograman.....	7
3.2. Kode Program.....	7
<b>4. Percobaan.....</b>	<b>11</b>
4.1. Kasus Invalid.....	11
4.1.1. Kasus 1.....	11
4.1.2. Kasus 2.....	11
4.1.3. Kasus 3.....	12
4.2. Kasus Solusi Ada.....	12
4.2.1. Kasus 1.....	12
4.2.2. Kasus 2.....	13
4.2.3. Kasus 3.....	13
4.3. Kasus Solusi Tidak Ada.....	14
4.3.1. Kasus 1.....	14
4.3.2. Kasus 2.....	14
<b>5. Lampiran.....</b>	<b>15</b>
<b>6. Referensi.....</b>	<b>16</b>
<b>7. Pernyataan.....</b>	<b>16</b>

## 1. Pendahuluan

### 1.1. Deskripsi Tugas

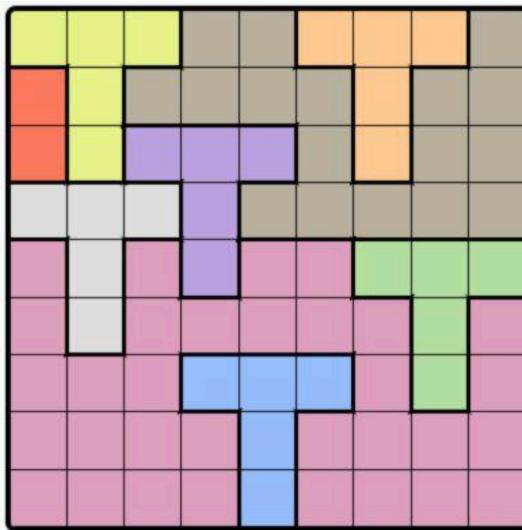


Queens adalah gim logika yang tersedia pada situs jejaring profesional LinkedIn. Tujuan dari gim ini adalah menempatkan queen pada sebuah papan persegi berwarna sehingga terdapat hanya satu queen pada tiap baris, kolom, dan daerah warna. Selain itu, satu queen tidak dapat ditempatkan bersebelahan dengan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

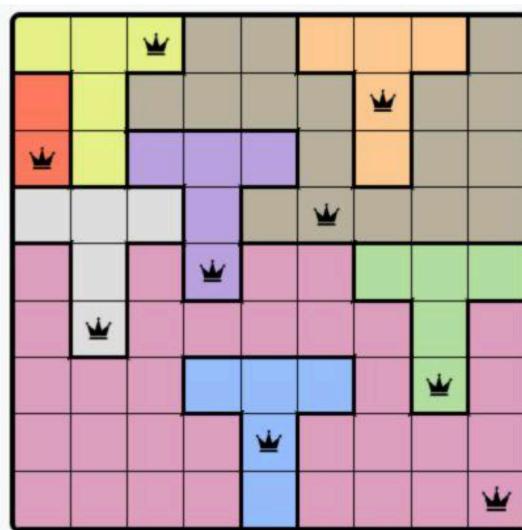
Tugas mahasiswa adalah membuat program yang dapat menemukan satu solusi penempatan queen pada suatu papan berwarna yang diberikan, atau menampilkan bahwa tidak ada solusi yang valid. Program melakukan pencarian solusi menggunakan algoritma brute force

## 1.2. Ilustrasi Kasus

Diberikan papan sebagai berikut. Untuk tugas ini, papan selalu dimulai kosong.



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



## 2. Pembahasan

### 2.1. Algoritma Brute Force

Algoritma brute force adalah suatu pendekatan yang lurus atau lempang untuk memecahkan suatu persoalan. Algoritma brute force memecahkan persoalan secara sangat sederhana, langsung, jelas caranya, serta mudah dipahami.

Contoh beberapa algoritma brute force juga digunakan pada permasalahan-permasalahan berikut:

- Menghitung pangkat
- Menghitung faktorial
- Mengalikan dua buah matriks, A dan B, berukuran
- Uji Bilangan Prima
- Pengurutan elemen
- Algoritma brute force pencocokan string
- Mencari Pasangan Titik yang Jaraknya Terdekat

Algoritma brute force umumnya tidak “cerdas” dan tidak sangkil, karena ia membutuhkan cost komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Kata “force” mengindikasikan “tenaga” ketimbang “otak”. Kadang-kadang algoritma brute force disebut juga algoritma naif (naïve algorithm). Algoritma brute force lebih cocok untuk persoalan yang ukuran masukannya ( $n$ ) kecil.

Kelebihan dari algoritma brute force antara lain adalah bahwa algoritma brute force dapat diterapkan untuk memecahkan hampir sebagian besar masalah. Selain itu, algoritma brute force sederhana dan mudah dimengerti, serta algoritma brute force menghasilkan algoritma bakuuntuk tugas-tugas komputasi seperti penjumlahan/perkalian buah bilangan, menentukan elemen minimum atau maksimum di dalam senarai (larik).

Di sisi lain, algoritma brute force jarang menghasilkan algoritma yang mangkus, umumnya lambat untuk masukan berukuran besar sehingga tidak dapat diterima, serta tidak sekontruktif/sekratif strategi pemecahan masalah lainnya.

### 2.2. Implementasi Algoritma Brute Force

Sebelum proses pencarian dimulai, algoritma membangun sebuah graf berukuran  $N^2 \times N^2$  untuk filterisasi tidak ada koordinat yang bersebelahan dan berada di region warna yang sama. Setiap sel pada papan ( $r,c$ ) dikonversi menjadi ID unik

menggunakan rumus  $(\text{baris} \times N) + \text{kolom}$  agar pencarian di graf menjadi lebih ringkas. Algoritma program memeriksa setiap pasang ID dalam graf. Jika dua ID memenuhi salah satu syarat: kedua sel memiliki karakter warna yang sama dalam data input atau kedua sel bersentuhan secara horizontal, vertikal, maupun diagonal, nilai dalam graf ditandai sebagai True (tidak memenuhi syarat). Dengan memindahkan logika jarak dan warna ke dalam graf di awal, proses validasi selama pencarian hanya memerlukan pengecekan nilai boolean sederhana.

Algoritma ini tidak memproses papan sebagai matriks dua dimensi selama tahap pencarian. Dalam algoritma ini, posisi ratu direpresentasikan dalam sebuah list angka berukuran  $N$ . Misalkan terdapat list  $p = [c_0, c_1, \dots, c_{N-1}]$ , indeks mewakili nomor baris dan nilai dalam indeks mewakili nomor kolom. Struktur data seperti ini secara otomatis memastikan hanya ada satu ratu tiap satu baris.

Selanjutnya, Algoritmanya adalah melakukan pemeriksaan secara menyeluruh terhadap setiap kemungkinan posisi ratu dengan tetap mematuhi ketentuan penempatan kolom yang telah ditetapkan. Algoritma menghasilkan setiap urutan angka dari  $[0, 0, \dots, 0]$  hingga  $[N-1, N-1, \dots, N-1]$ . Sebelum kombinasi angka di list diproses lebih jauh, dilakukan pengecekan apakah semua angka di dalam list tersebut bersifat unik. Jika semua angka unik, maka kandidat tersebut dianggap memenuhi syarat dasar satu ratu per baris dan kolom.

Terakhir, kombinasi koordinat yang dipilih kemudian dicek apakah setiap koordinat memiliki region warna yang berbeda dan apakah bersebelahan secara diagonal menggunakan graf yang telah dibuat sebelumnya. Algoritma program memeriksa nilai dua koordinat di graf. Jika bernilai True, kombinasi koordinat tersebut dinyatakan gagal karena melanggar aturan wilayah atau aturan kedekatan posisi. Jika semua kombinasi koordinat berhasil, sebuah solusi permainan papan ditemukan.

Algoritma ini merupakan algoritma brute force murni dan lebih tepatnya berbentuk exhaustive search karena algoritma ini mencoba seluruh kemungkinan penempatan ratu yang ada di papan permainan. Perbedaannya dengan algoritma brute force yang paling murni adalah algoritma ini tidak men-generate kemungkinan penempatan ratu pada baris yang sama dan menimbalasai kemungkinan filterisasi dengan graf melalui pemilihan kandidat koordinat pada proses sebelumnya. Hal ini tidak termasuk heuristik karena tidak ada prediksi dan optimisasi ini didasarkan pada aturan utama permainan ini yaitu satu ratu untuk tiap baris, kolom, dan warna.

### 2.3. Analisis Kompleksitas Waktu

Misalkan N adalah baris dan kolom pada grid permainan. Algoritma pencarian kolom dan baris mencoba setiap kemungkinan dari 0 sampai N-1 untuk setiap baris. Setiap baris memiliki kemungkinan N yaitu 0 sampai N-1 pilihan kolom sehingga kompleksitas waktunya adalah  $N^N$ . Sedangkan, algoritma filter region warna yang sama dan tes dua koordinat bersebelahan yang menggunakan graf menggunakan dua loop bersarang sehingga sederhananya kompleksitas waktunya adalah  $N^N$ . Secara teori, kompleksitas waktu untuk kasus terburuk (*worst case*) dalam algoritma ini adalah  $O(N^N \times N^2)$ .

## 3. Implementasi

### 3.1. Bahasa Pemrograman

Python adalah bahasa pemrograman yang digunakan dalam implementasi algoritma karena kesederhanaan sintaksisnya yang menyerupai bahasa manusia, sehingga memungkinkan fokus utama tetap pada pengembangan logika algoritma pencarian tanpa terhambat oleh kerumitan penulisan kode yang kaku.

### 3.2. Kode Program

```
import sys
import time
import string

grid = []
region = {}

filename = input("Insert txt name: ")

try:
    with open(filename, "r") as f:
        for line in f:
            row = list(line.strip())
            if row:
                grid.append(row)
        for char in row:
            if char not in string.ascii_uppercase:
```

```

        print(f"Invalid character '{char}'")
        sys.exit()

except FileNotFoundError:
    print("File not found.")
    sys.exit()

N = len(grid)

if not all(len(row) == N for row in grid):
    print("Grid isn't square.")
    sys.exit()

for r in range(N):
    for c in range(N):
        color = grid[r][c]
        id = (r * N) + c

        if color not in region:
            region[color] = []
            region[color].append(id)

if len(region) != N:
    print(f"Found {len(region)} region, expected {N}")
    sys.exit()

update = input("Enter how long each update in n-th iteration
(0 for none): ")

try:
    update = int(update) / 1000
    if update < 0: update = 0
except ValueError:
    update = 0

graph = [[False for _ in range(N*N)] for _ in range(N*N)]

for id1 in range(N*N):
    r1, c1 = id1 // N, id1 % N
    color1 = grid[r1][c1]

```

```

        for id2 in range(id1 + 1, N*N):
            r2, c2 = id2 // N, id2 % N
            color2 = grid[r2][c2]

            if color1 == color2:
                graph[id1][id2] = True
                graph[id2][id1] = True
                continue

            row_dist = r1 - r2 if r1 > r2 else r2 - r1
            col_dist = c1 - c2 if c1 > c2 else c2 - c1

            if row_dist <= 1 and col_dist <= 1:
                graph[id1][id2] = True
                graph[id2][id1] = True

p = [0] * N
iterations = 0
grid_solution = None
start = time.time()

while True:
    iterations += 1
    duplicate = False
    for i in range(N):
        for j in range(i + 1, N):
            if p[i] == p[j]:
                duplicate = True
                break
        if duplicate:
            break

    if not duplicate:
        valid = True
        for r1 in range(N):
            for r2 in range(r1 + 1, N):
                id1 = (r1 * N) + p[r1]
                id2 = (r2 * N) + p[r2]
                if graph[id1][id2]:

```

```
        valid = False
        break
    if not valid:
        break
    if valid:
        grid_solution = list(p)
        break

if (update > 0) and (iterations % update == 0):
    print(f"\nIterations: {iterations}")
    for r in range(N):
        row_str = ""
        for c in range(N):
            if p[r] == c:
                row_str += "# "
            else:
                row_str += grid[r][c] + " "
        print(row_str)
    print("\n\n")

d = N - 1
while d >= 0:
    p[d] += 1
    if p[d] < N:
        break
    else:
        p[d] = 0
        d -= 1
    if d < 0:
        break

duration = time.time() - start

if grid_solution:
    solution = "Solution\n"
    solution += "="*N + "\n"
    for r in range(N):
        row = ""
        for c in range(N):
```

```

        row += "# " if grid_solution[r] == c else
grid[r][c] + " "
        solution += row.rstrip() + "\n"
solution += "="*N
solution += f"\n\nTotal iterations:
{iterations}\nDuration: {duration:.4f} seconds\n"
else:
    solution = f"No solution found\nTotal Iterations:
{iterations}\nDuration: {duration:.4f} seconds"

print(solution)
save = input("\nDo you want to save the answer? (Y if want to
save): ")
if save == 'Y':
    path = input("Directory path: ")
    with open(path, "w") as f:
        f.write(solution)
    print(f"\nSaved to: {path}")
else:
    print("Answer isn't saved")

```

## 4. Percobaan

### 4.1. Kasus Invalid

#### 4.1.1. Kasus 1

AAAAAA  
ABCCBA  
ABD BD  
EFFBBD  
FEFBDD  
FFDDDD

Invalid character ''

#### 4.1.2. Kasus 2

```
AAAAAA  
ABCCBA  
ABDEBD  
EFFBBD  
FEFBDD
```

Grid isn't square.

#### 4.1.3. Kasus 3

```
AABBBC  
AABBBC  
AAEEECC  
EEEEEFF  
GGGEEFF  
GGGGFFF  
GGGGFFF
```

Found 6 region, expected 7

### 4.2. Kasus Solusi Ada

#### 4.2.1. Kasus 1

```
AABBBC  
AABBBC  
DDEEECC  
DDEEFF  
GGGEEFF  
GGGDDFF  
GGGDDFF
```

Solution

=====

```
# A B B B C C  
A A # B B C C  
D D E E E # C  
D D E # E F F  
G # G E E F F  
G G G D # F F  
G G G D D F #  
=====
```

Total iterations: 46732  
Duration: 0.0534 seconds

#### 4.2.2. Kasus 2

ABCCCCC  
BBBDEECC  
FBDDEECC  
FDDEEEEC  
FDEEEEEC  
FEEFFEEGC  
FEEFFEGGG  
FFFFFFFH  
  
Solution  
=====

# B C C C C C  
B B # D E E C C  
F B D D E E # C  
F D D E # E E C  
F # E E E E E C  
F E E # E E G C  
F E E F E # G G  
F F F F F F G #  
=====

Total iterations: 738032  
Duration: 0.9083 seconds

#### 4.2.3. Kasus 3

AAABBCCCD  
ABBBBCECD  
ABBBDCECD  
AAABDCCCD  
BBBBBDDDDD  
FGGGDDHDD  
FGIGDDHDD  
FGGGDDHHH

```
Solution  
=====
```

A A A B B C C # D  
A B B B # C E C D  
A B B B D C # C D  
A # A B D C C C D  
B B B B D # D D D  
F G G # D D H D D  
# G I G D D H D D  
F G # G D D H D D  
F G G G D D H H #  
=====

```
Total iterations: 323741637  
Duration: 517.4047 seconds
```

#### 4.3. Kasus Solusi Tidak Ada

##### 4.3.1. Kasus 1

```
ABA  
CAA  
CAA
```

```
No solution found  
Total Iterations: 27  
Duration: 0.0000 seconds
```

##### 4.3.2. Kasus 2

```
AA  
BB
```

```
No solution found  
Total Iterations: 4  
Duration: 0.0000 seconds
```

## 5. Lampiran

Link Github: [https://github.com/naufalromi/Tucil1\\_13524058](https://github.com/naufalromi/Tucil1_13524058)

Tabel Poin:

No	Poin	Ya	Tidak
1.	Program berhasil di kompilasi 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	✓	

## 6. Referensi

1. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Bruite-Force-\(2025\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Bruite-Force-(2025)-Bag1.pdf)
2. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/03-Algoritma-Bruite-Force-\(2025\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/03-Algoritma-Bruite-Force-(2025)-Bag2.pdf)

## 7. Pernyataan

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



Muhammad Naufal Romi Annafi