

Tugas Kecil IF2211 Strategi Algoritma

Penyelesaian Permainan Queens LinkedIn

Menggunakan Algoritma Brute Force



DAFTAR ISI

1. Algoritma Brute Force.....	3
a. Deskripsi Permasalahan.....	3
b. Pendekatan Algoritma.....	4
c. Langkah Langkah.....	4
2. Source Code.....	6
a. main.py.....	7
b. queensolver.py.....	9
c. validator.py.....	11
d. visual.py.....	12
e. boardreader.py.....	14
3. Test Case.....	16
a. Alur Program.....	17
b. Test board 4x4.....	18
c. Test board 5x5.....	19
d. Test board 6x6.....	19
e. Test board 9x9.....	20
f. Test board no solution.....	21
g. Test board invalid karakter.....	22
h. Test board invalid row/column.....	22
4. Repository.....	22

DFTAR GAMBAR

Gambar 1 code main.py.....	8
Gambar 2 code queensolver.py.....	10
Gambar 3 code validator.py.....	11
Gambar 4 code visual.py.....	13
Gambar 5 code boardreader.py.....	16
Gambar 6 live update iterasi.....	17
Gambar 6 Pemilihan penyimpanan.....	18
Gambar 7 Pemilihan File.....	18
Gambar 8 Solusi 4x4 ditemukan.....	18
Gambar 9 Solusi 5x5 ditemukan.....	19
Gambar 10 Solusi 6x6 ditemukan.....	20
Gambar 11 Solusi 9x9 ditemukan.....	21
Gambar 12 tidak ada solusi ditemukan.....	21
Gambar 13 jika board tidak valid.....	22
Gambar 14 jika invalid ro/coloum.....	22

1. Algoritma Brute Force

a. Deskripsi Permasalahan

Permainan queens linkedIn adalah permainan logika yang tersedia di LinkedIn dengan aturan sebagai berikut:

- Setiap baris harus memiliki tepat satu queen
- Setiap kolom harus memiliki tepat satu queen
- Setiap region warna harus memiliki tepat satu queen
- Tidak boleh ada dua queen yang bersebelahan (termasuk diagonal 1 langkah)

Program ini menyelesaikan permainan Queens menggunakan algoritma brute force murni (exhaustive search) tanpa optimasi heuristik apapun.

b. Pendekatan Algoritma

Algoritma brute force yang diimplementasikan menggunakan pendekatan generate-and-test. Secara umum, algoritma bekerja dengan langkah-langkah berikut:

- **Generate:** Membangkitkan semua kemungkinan permutasi penempatan queen (satu queen per baris)
- **Test:** Mengecek validitas setiap permutasi berdasarkan aturan permainan
- **Return:** Mengembalikan permutasi pertama yang valid sebagai solusi

Solusi direpresentasikan sebagai permutasi dari angka 0 sampai dengan $n-1$ yang dimana indeks array akan mempresentasikan baris dan nilai pada indeks tersebut tersebut mempresentasikan kolom tempat **queen** berada

c. Langkah Langkah

i. Inisialisasi

Sebelum memulai proses pencarian solusi, program akan melakukan validasi input yang diterima untuk memastikan data yang dijalankan sudah sesuai spesifikasi. Validasi validasi yang dilakukan adalah:

- Membaca File Input
Program membaca file teks (.txt) yang berisi konfigurasi papan, dimana setiap karakter merepresentasikan warna region.
- Validasi Karakter
Semua karakter harus berupa huruf alfabet (A-Z). Jika ditemukan karakter selain huruf, program akan menampilkan pesan error dan berhenti.
- Validasi Bentuk Papan
Papan harus berbentuk persegi ($n \times n$). Jumlah baris harus sama dengan panjang setiap baris.
- Validasi Jumlah Region
Program menghitung jumlah warna unik pada papan. Jumlah region warna harus sama dengan ukuran papan (n). Jika tidak sama, input dianggap invalid.

Validasi ini penting untuk menghindari error runtime dan memastikan bahwa masalah yang akan diselesaikan memiliki struktur yang benar.

ii. Generate Permutasi

Program ini membangkitkan semua permutasi dengan cara:

- Mulai dengan permutasi awal [0, 1, 2, ..., n-1]
- Gunakan array counter c[] untuk tracking posisi swap
- Untuk setiap posisi i:
 - Jika posisi $c[i] < 1$
 - Jika i genap = swap arr[0] dengan arr[i]
 - Jika i ganjil = swap arr[c[i]] dengan arr[i]
 - Jika posisi $c[i] \geq 0$
 - Reset $c[i]$ ke 0, increment i
- Ulangi semua sampai $n!$ Permutassu dibangkitkan

Untuk implementasi pada kodennya adalah berikut:

```
def permutations(self, arr):  
    arr = list(arr)  
    n = len(arr)  
    c = [0] * n  
  
    yield tuple(arr)  
  
    i = 0  
    while i < n:  
        if c[i] < i:  
            if i % 2 == 0:  
                arr[0], arr[i] = arr[i], arr[0]  
            else:  
                arr[c[i]], arr[i] = arr[i], arr[c[i]]  
            yield tuple(arr)  
            c[i] += 1  
            i = 0  
        else:  
            c[i] = 0  
            i += 1
```

iii. Validasi

1. Validasi warna

Setelah sebuah permutasi dibangkitkan, langkah pertama validasi adalah memeriksa apakah setiap queen berada di region warna yang berbeda. Constraint ini harus dipenuhi karena merupakan salah satu aturan utama dalam Queens LinkedIn. Untuk implementasinya adalah sebagai berikut:

```
def check_color_constraints(self, permutation):  
    color_used = {}
```

```

        for row in range (self.n):
            col = permutation[row]
            color = self.board[row][col]

            if color in color_used:
                return False

            color_used[color] = True

        return True
    
```

Jadinya untuk setiap baris queens permutassi akan dicek setiap warnanya jika ada yang berwarna sama maka akan return false dan lanjut ke posisi queens berikutnya, jika sudah ketemu program akan berhenti dan akan return true, jika sampai posisi queen terakhir tidak ditemukan solusinya, maa program akan mereturn false

2. Validasi tetangga

Setelah constraint warna terpenuhi, program memeriksa apakah ada pasangan queen yang bersebelahan. Dua queen dianggap bersebelahan jika jaraknya pada sumbu horizontal DAN vertikal masing-masing ≤ 1 (termasuk diagonal). Untuk implementasinya adalah sebagai berikut:

```

def check_neighbor(self, permutation):
    queens_positions = [(row, permutation[row]) for row
in range(self.n)]

    for i in range(len(queens_positions)):
        for j in range(i + 1, len(queens_positions)):
            r1, c1 = queens_positions[i]
            r2, c2 = queens_positions[j]
            if abs(r1 - r2) <= 1 and abs(c1 - c2) <= 1:
                return False

    return True
    
```

Pada program ini permutasi akan di konversi menjadi list koordinat dan akan menggunakan nested loop untuk mengecek semua pasangan. Jika terdeteksi terdapat pelanggaran maka akan lanjut ke iterasi berikutnya.

iv. Return Solusi

Setelah kedua validasi (warna dan adjacency) selesai, program akan mengambil keputusan:

```
if validator.is_valid(permutation):
```

```
        end_time = time.time()
        self.time_taken = (end_time - start_time) *
1000
        self.solution = permutation
    return permutation
```

Jika semua $n!$ permutasi sudah dicoba dan tidak ada yang valid, generator akan habis (StopIteration) dan program mengembalikan None, menandakan bahwa tidak ada solusi untuk konfigurasi papan tersebut.

2. Source Code

Program ini diimplementasikan dalam bahasa python dengan struktur modular yang memisahkan tanggung jawab setiap komponen. Pendekatan modular ini memberikan keuntungan dalam hal maintainability, testability, dan code reusability struktur programnya sebagai berikut:

```
src/
├── main.py
├── queensolver.py
├── validator.py
├── boardreader.py
└── visual.py
```

a. main.py

File utama yang mengatur alur eksekusi program dari awal hingga akhir. File ini bertindak sebagai orchestrator yang mengoordinasikan interaksi antar modul.

```
1 import os
2 import sys
3 from visual import Visual
4 from boardreader import BoardReader
5 from validator import Validator
6 from queensolver import QueenSolver
7
8 def print_header():
9     print("\n" + "="*60)
10    print(" " * 15 + "QUEENS SOLVER")
11    print(" " * 10 + "Brute Force Algorithm")
12    print("="*60 + "\n")
13
14 def get_input_file():
15     print("File yang tersedia:")
16
17     test_dir = "../test"
18     if os.path.exists(test_dir):
19         test_file = [f for f in os.listdir(test_dir) if f.endswith('.txt')]
20
21         if test_file:
22             print("\nFile yang tersedia")
23             for i, file in enumerate(test_file, 1):
24                 print(f"{i}. {file}")
25             print()
26
27     while True:
28         filename = input("Masukan nama file ").strip()
29         if not os.path.exists(filename):
30             test_path = os.path.join(test_dir, filename)
31             if os.path.exists(test_path):
32                 filename = test_path
33             else:
34                 print ("File tidak ditemukan.")
35             continue
36
37     return filename
38
39
40
41 def main():
42     try:
43         print_header()
44
45         filename = get_input_file()
46         print(f"Memuat papan dari {filename}...")
47         reader = BoardReader()
48         board = reader.read_file(filename)
49         reader.validate_board()
50         validator = Validator(board)
51         visual = Visual(board)
52         queen_solver = QueenSolver(board)
53         validator.is_region_valid()
54
55         print("\nPapan yang dimuat:")
56         visual.display_board()
57
58         input("Tekan Enter untuk memulai pencarian solusi...")
59         solution = queen_solver.solve(validator, visual)
60         stats = queen_solver.get_statistics()
61
62         if solution:
63             visual.display_solution(solution, stats['time_ms'], stats['iterations'])
64             save = input("Apakah Anda ingin menyimpan solusi? (y/n): ").strip().lower()
65             if save == 'y':
66                 save_filename = input("Masukan nama file untuk menyimpan solusi: ").strip()
67
68                 if not save_filename:
69                     save_filename = "solusi.txt"
70
71                 if not save_filename.endswith('.txt'):
72                     save_filename += '.txt'
73
74                 output_dir = "../output"
75                 if not os.path.exists(output_dir):
76                     os.makedirs(output_dir)
77
78                 output_path = os.path.join(output_dir, save_filename)
79                 queen_solver.save_solution(output_path, solution)
80
81
82         else:
83             visual.print_no_solution(stats['time_ms'], stats['iterations'])
84
85         print("Program selesai.")
86
87     except FileNotFoundError as e:
88         print(e)
89         sys.exit(1)
90     except ValueError as e:
91         print(e)
92         sys.exit(1)
93
94 if __name__ == "__main__":
95     main()
```

Gambar 1 code main.py

File main.py berfungsi sebagai entry point yang mengatur alur program dari input hingga output. Program meminta user memilih file, membaca dan memvalidasi papan, menginisialisasi objek-objek yang diperlukan (Validator, Visual, QueenSolver), menjalankan pencarian solusi, dan menampilkan hasil beserta statistik waktu dan iterasi.

b. queensolver.py

Modul ini berisi implementasi inti dari algoritma brute force, termasuk generator permutasi pencarian solusi.

```
1 import time
2 import math
3
4 class QueenSolver:
5     def __init__(self, board):
6         self.board = board
7         self.n = len(board)
8         self.iteration_count = 0
9         self.solution = None
10        self.time_taken = 0
11
12    def permutations(self, arr):
13        arr = list(arr)
14        n = len(arr)
15        c = [0] * n
16
17        yield tuple(arr)
18
19        i = 0
20        while i < n:
21            if c[i] < i:
22                if i % 2 == 0:
23                    arr[0], arr[i] = arr[i], arr[0]
24                else:
25                    arr[c[i]], arr[i] = arr[i], arr[c[i]]
26                yield tuple(arr)
27                c[i] += 1
28                i = 0
29            else:
30                c[i] = 0
31                i += 1
32
33    def solve(self, validator, visualizer=None):
34        print("Mencari solusi...")
35
36        start_time = time.time()
37
38        if self.n <= 4:
39            interval = 1
40        elif self.n <= 7:
41            interval = 10
42        else:
43            interval = 10000
44
45        total_permutations = math.factorial(self.n)
46        initial = list(range(self.n))
47
48        for permutation in self.permutations(initial):
49            self.iteration_count += 1
50
51            if visualizer and self.iteration_count % interval == 0:
52                visualizer.animate_progress(permutation, self.iteration_count, total_permutations)
53
54            if validator.is_valid(permutation):
55                end_time = time.time()
56                self.time_taken = (end_time - start_time) * 1000
57                self.solution = permutation
58
59        return permutation
60
61    end_time = time.time()
62    self.time_taken = (end_time - start_time) * 1000
63    print()
64    return None
65
66    def get_statistics(self):
67        return {
68            'time_ms': self.time_taken,
69            'iterations': self.iteration_count,
70            'solution': self.solution
71        }
72
73    def save_solution(self, filename, permutation):
74        try:
75            with open(filename, 'w') as f:
76                for row in range(self.n):
77                    col = permutation[row]
78                    line = ""
79                    for c in range(self.n):
80                        if c == col:
81                            line += "#"
82                        else:
83                            line += self.board[row][c]
84                    f.write(line + "\n")
85
86            print(f"Solusi disimpan ke {filename}")
87            return True
88
89        except Exception as e:
90            print(f"Galat menyimpan solusi ke {e}")
91            return False
92
93
```

Gambar 2 code queensolver.py

Class QueenSolver mengimplementasikan algoritma brute force dengan Heap's Algorithm. Fungsi permutations() membangkitkan semua $n!$ permutasi secara lazy, sedangkan fungsi solve() mencoba setiap permutasi dan memvalidasinya. Jika

ditemukan permutasi valid, solusi langsung dikembalikan (early termination). Program menampilkan live update pada interval tertentu untuk feedback visual.

c. validator.py

Modul ini berisi logic untuk memvalidasi setiap permutasi terhadap constraint warna dan adjacency.

```
● ● ●
1  class Validator:
2      def __init__(self, board):
3          self.board = board
4          self.n = len(board)
5          self.color_regions = self.build_color_map()
6
7      def build_color_map(self):
8          color_map = {}
9
10         for i in range(self.n):
11             for j in range (self.n):
12                 color = self.board[i][j]
13                 if color not in color_map:
14                     color_map[color] = []
15                     color_map[color].append((i,j))
16
17         return color_map
18
19     def is_region_valid(self):
20         num_colors = len(self.color_regions)
21         if num_colors != self.n:
22             raise ValueError(f"Jumlah warna tidak sama dengan ukuran papan, \nPapan = {self.n} \nWarna = {num_colors}")
23
24     def is_valid(self, permutation):
25         if not self.check_color_cinstraints(permutation):
26             return False
27
28         if not self.check_neighbor(permutation):
29             return False
30
31         return True
32
33     def check_color_cinstraints(self, permutation):
34         color_used = {}
35
36         for row in range (self.n):
37             col = permutation[row]
38             color = self.board[row][col]
39
40             if color in color_used:
41                 return False
42
43             color_used[color] = True
44
45         return True
46
47     def check_neighbor(self, permutation):
48         queens_positions = [(row, permutation[row]) for row in range(self.n)]
49
50         for i in range(len(queens_positions)):
51             for j in range(i + 1, len(queens_positions)):
52                 r1, c1 = queens_positions[i]
53                 r2, c2 = queens_positions[j]
54                 if abs(r1 - r2) <= 1 and abs(c1 - c2) <= 1:
55                     return False
56
57         return True
58
```

Gambar 3 code validator.py

Class Validator memvalidasi setiap permutasi terhadap constraint warna dan adjacency. Fungsi check_color_constraints() memastikan tidak ada dua queen di warna yang sama menggunakan dictionary ($O(n)$), sedangkan check_neighbor() mengecek semua pasangan queen untuk memastikan tidak bersebelahan ($O(n^2)$). Validasi warna dicek terlebih dahulu untuk efisiensi (short-circuit).

d. visual.py

Modul untuk menampilkan visualisasi papan ke terminal dan memberikan live update selama proses pencarian berlangsung.



```

1 import os
2 import sys
3
4 class Visual:
5     def __init__(self, board):
6         self.board = board
7         self.n = len(board)
8
9     def clear_screen(self):
10        os.system('cls' if os.name == 'nt' else 'clear')
11
12    def display_board(self, permutation = None, clear = None):
13        if clear:
14            self.clear_screen()
15            print ("\n" + "*40)
16
17        for row in range (self.n):
18            line = ""
19            for col in range (self.n):
20                has_queen = False
21                if permutation:
22                    has_queen = (permutation[row] == col)
23                if has_queen:
24                    line += "# "
25                else:
26                    line += self.board[row][col] + " "
27            print(line)
28
29        print("*40 + "\n")
30
31
32
33    def display_solution(self, permutation, time_ms, iteration):
34        print("\nSolusi Ditemukan\n")
35
36        self.display_board(permutation)
37
38        print(f"Waktu pencarian: {time_ms:.2f} ms")
39        print(f"Iterasi: {iteration}")
40
41    def print_no_solution(self, time_ms, iteration):
42        print("Tidak ada solusi yang ditemukan")
43        print(f"Waktu pencarian: {time_ms:.2f} ms")
44        print(f"Iterasi: {iteration}")
45
46    def animate_progress(self, permutation, iteration, total):
47        sys.stdout.write(f"\riterasi: {iteration}/{total}")
48        sys.stdout.flush()
49
50        print("\n" + "*40)
51        for row in range (self.n):
52            line = ""
53            for col in range (self.n):
54                has_queen = (permutation[row] == col)
55                if has_queen:
56                    line += "# "
57                else:
58                    line += self.board[row][col] + " "
59            print(line)
60        print("*40 + "\n")
61

```

Gambar 4 code visual.py

Class Visual menampilkan papan dan feedback visual ke terminal. Fungsi display_board() menampilkan papan dengan simbol '#' untuk queen dan huruf untuk

warna region. Fungsi `animate_progress()` memberikan live update non-blocking menggunakan `sys.stdout.write()` tanpa `time.sleep()` sehingga tidak memperlambat pencarian.

e. `boardreader.py`

Modul untuk membaca file input dan melakukan validasi format papan.

```
● ● ●
```

```
1  class BoardReader:
2      def __init__(self):
3          self.board = []
4          self.n = 0
5
6      def read_file(self, filename):
7          try:
8              with open (filename, 'r') as f:
9                  lines = f.readlines()
10
11             self.board = []
12             for line in lines:
13                 line = line.strip()
14                 if line :
15                     self.board.append(list(line))
16
17             if not self.board:
18                 raise ValueError("File Tidak Valid")
19
20             self.n = len(self.board)
21
22         return self.board
23
24     except FileNotFoundError:
25         raise FileNotFoundError(f"file'{filename}' tidak ditemukan")
26     except Exception as e:
27         raise e
28
29
30     def validate_board(self):
31         for i, row in enumerate(self.board):
32             for j, cell in enumerate(row):
33                 if not cell.isalpha():
34                     raise ValueError(
35                         f"Karakter '{cell}' pada posisi ({i},{j}) tidak valid "
36                         f"Karakter harus berupa alphabet"
37                     )
38
39         row_lengths = [len(row) for row in self.board]
40         if len(set(row_lengths)) != 1:
41             raise ValueError("Panjang baris tidak konsisten")
42
43         if row_lengths[0] != self.n:
44             raise ValueError("Panjang baris tidak sesuai dengan ukuran papan")
45
46         print("board valid")
47
48     def get_board(self):
49         return self.board
50
51     def get_size(self):
52         return self.n
```

Gambar 5 code boardreader.py

Class BoardReader membaca file input dan memvalidasi format papan. Fungsi `read_file()` mem-parsing file menjadi matriks 2D, sedangkan `validate_board()` memastikan semua karakter adalah huruf (A-Z), semua baris memiliki panjang sama, dan papan berbentuk persegi. Error handling diterapkan untuk `FileNotFoundException` dan `ValueError`.

3. Test Case

Berikut adalah contoh eksekusi program dengan berbagai ukuran papan dan tingkat kesulitan untuk mendemonstrasikan kemampuan dan keterbatasan algoritma brute force. Program dilengkapi dengan fitur live update yang menampilkan proses pencarian solusi secara real-time, memungkinkan pengguna untuk melihat bagaimana algoritma brute force bekerja dengan menampilkan konfigurasi papan yang sedang diuji pada setiap interval tertentu.

Program tidak menampilkan setiap permutasi karena akan terlalu cepat dan membingungkan, melainkan menampilkan update setiap N iterasi dimana interval ditentukan berdasarkan ukuran papan. Untuk papan 4x4 atau lebih kecil update dilakukan setiap 1 iterasi, papan 5x5 sampai 7x7 setiap 10 iterasi, dan papan 8x8 atau lebih besar setiap 10,000 iterasi.

```
Mencari solusi...
iterasi: 1/24
=====
# A A B
A # B B
C C # B
C C D #
=====

iterasi: 2/24
=====
A # A B
# A B B
C C # B
C C D #
=====

iterasi: 3/24
=====
A A # B
# A B B
C # B B
C C D #
=====
```

Gambar 6 live update iterasi

a. Alur Program

Sebelum menjalankan setiap test case, pengguna terlebih dahulu harus memilih file input yang diinginkan dari folder test. Program akan menampilkan daftar file yang tersedia dan meminta user untuk memilih nomor file atau memasukkan path secara manual. Setelah file dipilih, program akan membaca konfigurasi papan, memvalidasi input, dan memulai proses pencarian solusi menggunakan algoritma brute force. Selama proses pencarian berlangsung, program menampilkan live update yang

menunjukkan iterasi saat ini dan konfigurasi papan yang sedang dicoba. Ketika solusi ditemukan atau semua permutasi sudah dicoba, program menampilkan hasil akhir berupa papan dengan posisi queen (ditandai '#'), waktu pencarian dalam milidetik, dan jumlah iterasi yang dilakukan. Setelah hasil ditampilkan, program memberikan opsi kepada user untuk menyimpan hasil ke file output. Jika user memilih 'y', hasil akan disimpan dalam folder output dengan format nama file yang mencantumkan timestamp, sehingga setiap hasil test case tersimpan dengan rapi dan tidak saling tertimpa.

The image shows two terminal windows side-by-side. The left window displays the found solution for a 4x4 board:

```
Solusi Ditemukan
=====
A # A B
A A B #
# C B B
C C # D
=====

Waktu pencarian: 2.03 ms
Iterasi: 8
Apakah Anda ingin menyimpan solusi? (y/n):
```

The right window shows the file selection menu:

```
=====
QUEENS SOLVER
Brute Force Algorithm
=====

File yang Tersedia:
File yang tersedia
1. test2x2ns.txt
2. test4x4.txt
3. test5x5.txt
4. test6x6.txt
5. test7x8.txt
6. test9x9.txt

Masukan nama file:
```

Gambar 6 Pemilihan penyimpanan

Gambar 7 Pemilihan File

b. Test board 4x4

The image shows a single terminal window displaying the found solution for a 4x4 board:

```
Solusi Ditemukan
=====
A # A B
A A B #
# C B B
C C # D
=====

Waktu pencarian: 2.03 ms
Iterasi: 8
```

Gambar 8 Solusi 4x4 ditemukan

c. Test board 5x5

```
Solusi Ditemukan
=====
A # A B C
A B B # C
# B B E C
D D E E #
D D # E E
=====

Waktu pencarian: 1.33 ms
Iterasi: 49
```

Gambar 9 Solusi 5x5 ditemukan

d. Test board 6x6

Solusi Ditemukan

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

Waktu pencarian: 1.10 ms

Iterasi: 35

Gambar 10 Solusi 6x6 ditemukan

e. Test board 9x9

Solusi Ditemukan

```
=====
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 #
=====
```

Waktu pencarian: 10.30 ms

Iterasi: 15601

Gambar 11 Solusi 9x9 ditemukan

f. Test board no solution

```
Tidak ada solusi yang ditemukan
Waktu pencarian: 0.00 ms
Iterasi: 2
Program selesai.
```

Gambar 12 tidak ada solusi ditemukan

g. Test board invalid karakter

```
Masukan nama file test7x8.txt  
Memuat papan dari ../test\test7x8.txt...  
Karakter '1' pada posisi (0,6) tidak valid Karakter harus berupa alphabet
```

Gambar 13 jika board tidak valid

h. Test board invalid row/column

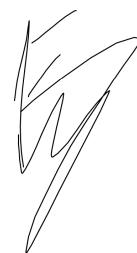
```
Jumlah warna tidak sama dengan ukuran papan,  
Papan = 7  
Warna = 8
```

Gambar 14 jika invalid ro/coloum

4. Repository

Link Repo: <https://github.com/FudhailFayyadh/TucilBasmalah/tree/main/test>

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



Fudhail Fayyadh (18223121)

No	Poin	ya	tidak
	Program berhasil di kompilasi tanpa kesalahan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	Program berhasil di jalankan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	Solusi yang diberikan program benar dan mematuhi aturan permainan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	Program memiliki Graphical User Interface (GUI)	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	Program dapat menyimpan solusi dalam bentuk file gambar	<input type="checkbox"/>	<input checked="" type="checkbox"/>