Laporan Tugas Kecil 1 IF2211 Strategi Algoritma Semester II Tahun 2025/2026

Penyelesaian Permainan Queens Linkedin

Disusun Oleh

Beni Lesmana 10122043

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

A. DESKRIPSI ALGORITMA

Algoritma yang diimplementasikan untuk menyelesaikan masalah Queens ini adalah pendekatan Brute Force Exhaustive Search dengan Backtracking. Algoritma ini bekerja dengan cara mengeksplorasi ruang solusi secara sistematis untuk menempatkan N ratu pada papan berukuran N x N. Langkah-langkah detail algoritma adalah sebagai berikut:

1. **Inisialisasi & Identifikasi Region**: Program membaca papan permainan dan memetakan setiap sel ke dalam suatu kelompok warna (region). Hasilnya disimpan dalam struktur data dictionary atau hash map untuk akses cepat **O(1)** saat validasi.

2. **Pencarian Rekursif (*Row-by-Row*)**: Pencarian dimulai dari baris ke-0. Pada setiap baris, algoritma mencoba menempatkan ratu di setiap kolom (dari indeks 0 hingga N-1).

3. **Validasi Langkah (*Pruning*)**: Sebelum ratu ditempatkan secara permanen pada posisi (row, col), dilakukan pemeriksaan validasi terhadap tiga aturan utama :

    a. Aturan Baris & Kolom: Memastikan tidak ada ratu lain di baris atau kolom yang sama.

    b. Aturan Region Warna: Memastikan warna pada sel (row, col) belum ditempati oleh ratu lain yang sudah diletakkan sebelumnya.

    c. Aturan Ketetanggaaan (Neighbors): Memastikan ratu tidak bersentuhan dengan ratu lain secara vertikal, horizontal, maupun diagonal.

4. **Rekursi & *Backtracking***:

    a. Jika posisi valid, ratu akan ditempatkan dan fungsi memanggil dirinya sendiri secara rekursif untuk baris berikutnya (row + 1) .

    b. Jika fungsi rekursif mengembalikan True, berarti solusi ditemukan.

    c. Jika seluruh kolom pada baris tersebut telah dicoba dan tidak ada yang valid (atau panggilan rekursif mengembalikan False), algoritma akan melakukan *backtracking* dengan membatalkan penempatan ratu terakhir dan mencoba kolom berikutnya.

5. **Terminasi**: Algoritma berhenti ketika:

    a. Semua ratu berhasil ditempatkan (Solusi ditemukan).

    b. Seluruh kemungkinan pada baris awal telah dicoba (Tidak ada solusi).

B. SOURCE CODE

Algoritma yang dibangun menggunakan bahasa pemrograman python dengan modul-modul sebagai berikut:

1. **solver.py**

   Berisi kelas board_solver yang berisi fungsi-fungsi yang berguna untuk menyelesaikan masalah menggunakan strategi brute force, mulai dari mapping warna, validasi posisi, dan rekursifnya. Berikut adalah kodenya :

```
import time


class board_solver :
    def __init__(self, board:list[list[str]]):
        self.board = board
        self.n = len(board)
        self.solution = [[' ' for i in range(self.n)] for j in range(self.n)]
        self.iterations = 0
        self.start_time = 0
        self.execution_time = 0
        self.regions = self.color_regions()
        self.visualization_call = None

    # Mapping setiap warna ke posisi sel nya
    def color_regions(self) :
        regions = {}
        for i in range(self.n) :
            for j in range(self.n) :
                color = self.board[i][j]
                if color not in regions :
                    regions[color] = []
                regions[color].append((i,j))
        return regions

    # Cek setiap pasangan bidak apakah bertetangga
    def check_neighbor (self, row1: int, col1: int, row2: int, col2: int) :
        return abs(row1-row2) <= 1 and abs(col1-col2) <= 1 and (row1!=row2 or
col1!=col2)

    # Cek apakah posisi bidak valid
    def check_valid(self, row: int, col: int, placed_queens: list[tuple[int,int]]) :
        self.iterations +=1
        current_color = self.board[row][col]
        for qr, qc in placed_queens :
            if qr == row :
                return False
            elif qc == col :
                return False
            elif self.board[qr][qc] == current_color :
                return False
            elif self.check_neighbor(row,col,qr,qc) :
                return False
        return True

    # Pemasangan bidak secara rekursif
    def solver_recursive(self, row: int, placed_queens: list[tuple[int,int]]) :
        if row == self.n :
            return True

        for col in range(self.n) :
            if self.check_valid(row, col, placed_queens) :
                placed_queens.append((row,col))

                if self.visualization_call:
                    self.visualization_call(placed_queens, self.iterations)

                if self.solver_recursive(row+1, placed_queens) :
                    return True

                placed_queens.pop()

        return False

    # Fungsi utama untuk Solve
    def solve(self, visualization_call=None) :
        self.iterations = 0
        self.start_time = time.time()
        placed_queens = []
        self.visualization_call = visualization_call

        success = self.solver_recursive(0,placed_queens)
        self.execution_time = (time.time() - self.start_time) * 1000

        if success :
            for qr, qc in placed_queens:
                self.solution[qr][qc] = '#'

            for i in range (self.n) :
                for j in range (self.n) :
                    if self.solution[i][j]!='#' :
                        self.solution[i][j] = self.board[i][j]

        return success, self.solution, self.iterations, self.execution_time,
placed_queens
```

2. **utils.py**

Berisi fungsi dan kelas yang berguna untuk visualisasi GUI dan menyimpan hasil baik text maupun gambar. Berikut adalah kodenya :

```
 import os
import tkinter as tk
from tkinter import filedialog, messagebox
```

```python
from PIL import Image, ImageDraw, ImageFont

color_palette = [
    '#FF6B6B',
    '#FFE66D',
    '#4ECDC4',
    '#95E1D3',
    '#A8E6CF',
    '#FFB6B9',
    '#C7CEEA',
    '#B4A7D6',
    '#FFDAC1',
    '#FF8B94',
    '#6C5CE7',
    '#00B894',
    '#FDCB6E',
    '#E17055',
    '#74B9FF',
    '#A29BFE',
    '#FD79A8',
    '#DCDDE1',
    '#00CEC9',
    '#FFEAA7',
    ]

# Fungsi untuk membaca file dan membangun papan
def read_file(filename: str) :
    if not os.path.exists(filename) :
        raise FileNotFoundError(f"File '{filename}' not found!")

    with open(filename, 'r') as f :
        lines = f.readlines()

    board = []
    for line in lines:
        line = line.strip()
        if line :
            board.append(list(line))

    if not board :
        raise ValueError('Board is empty!')

    n = len(board)
    for row in board :
        if len(row)!=n:
            raise ValueError("Board must be square (NxN)!")

    return board

# Fungsi untuk menyimpan solusi
def save_text(solution: list[list[str]], filename: str) :
    with open(filename, 'w') as f :
        for row in solution :
            f.write(''.join(row) + '\n')

# Fungsi untuk mengambil warna
def get_color(char: str, map: dict) :
    if char not in map :
        map[char] = len(map) % len(color_palette)

    return color_palette[map[char]]

# Fungsi untuk menyimpan gambar
def save_image(board: list[list[str]], queens: list[tuple[int,int]], filename: str,
cell_size: int=60):
    n = len(board)
    img_size = n*cell_size
    img = Image.new('RGB', (img_size, img_size), color="#000000")
    draw = ImageDraw.Draw(img)
    color_map = {}

    for i in range (n) :
        for j in range (n) :
            x1 = j*cell_size
            y1 = i*cell_size
            x2 = x1 + cell_size
            y2 = y1 + cell_size

            cell_color = get_color(board[i][j], color_map)
            draw.rectangle([x1, y1, x2, y2], fill=cell_color, outline='#34495E',
width=2)

    font = ImageFont.truetype("seguisym.ttf", int(cell_size * 0.5))
    for qr, qc in queens :
        x = qc*cell_size + cell_size//2
        y = qr*cell_size + cell_size//2

        crown = '♕'

        draw.text((x+2, y+2), crown, fill="#FFFEFE", font=font, anchor="mm" )
        draw.text((x, y), crown, fill="#000000", font=font, anchor="mm")

    try:
        img.save(filename, 'PNG')
        return True
    except Exception as e:
        print(f"Error saving image: {e}")
        return False

# Class untuk bonus GUI
class GUI :
    def __init__(self, board: list[list[str]], solver_call: callable):
        self.board = board
        self.n = len(board)
        self.solver_call = solver_call
        self.cell_size = min(60, 600//self.n)
        self.root = tk.Tk()
        self.root.title("Brute Force Algorithm for Queens Linkedin Problem")
        self.root.configure(bg='#2C3E50')

        self.color_map = {}
        self.canvas_frame = tk.Frame(self.root, bg='#2C3E50')
        self.canvas_frame.pack(pady=20)
        self.canvas = tk.Canvas(
            self.canvas_frame,
            width=self.n*self.cell_size,
            height=self.n*self.cell_size,
            bg='#34495E',
            highlightthickness=0)
        self.canvas.pack()

        self.info_frame = tk.Frame(self.root, bg='#2C3E50')
        self.info_frame.pack(pady=10)

        self.iteration_label = tk.Label(
            self.info_frame,
            text="Iteration: 0",
            font=('Arial', 14, 'bold'),
            bg='#2C3E50',
            fg='#ECF0F1')
        self.iteration_label.pack()

        self.time_label = tk.Label(
            self.info_frame,
            text="Time: 0 ms",
            font=('Arial', 12),
            bg='#2C3E50',
            fg='#BDC3C7'
        )
        self.time_label.pack()

        self.button_frame = tk.Frame(self.root, bg='#2C3E50')
        self.button_frame.pack(pady=10)

        self.solve_button = tk.Button(
            self.button_frame,
            text="Start",
            command=self.start_solving,
            font=('Arial', 12, 'bold'),
            bg='#27AE60',
            fg='white',
            padx=20,
            pady=10,
            relief=tk.RAISED,
            cursor='hand2'
        )
        self.solve_button.pack(side=tk.LEFT, padx=5)
```

```python
        self.save_image_button = tk.Button(
            self.button_frame,
            text="Save Image",
            command=self.save_as_image,
            font=('Arial', 12),
            bg='#3498DB',
            fg='white',
            padx=20,
            pady=10,
            relief=tk.RAISED,
            cursor='hand2',
            state=tk.DISABLED
        )
        self.save_image_button.pack(side=tk.LEFT, padx=5)

        self.save_text_button = tk.Button(
            self.button_frame,
            text="Save Text",
            command=self.save_as_text,
            font=('Arial', 12),
            bg='#E67E22',
            fg='white',
            padx=20,
            pady=10,
            relief=tk.RAISED,
            cursor='hand2',
            state=tk.DISABLED
        )
        self.save_text_button.pack(side=tk.LEFT, padx=5)

        self.queens = []
        self.solution = None
        self.execution_time = 0
        self.draw_board()

    def draw_board(self, queens: list[tuple[int,int]]=None) :
        self.canvas.delete('all')

        for i in range (self.n) :
            for j in range (self.n) :
                x1 = j * self.cell_size
                y1 = i * self.cell_size
                x2 = x1 + self.cell_size
                y2 = y1 + self.cell_size

                cell_color = get_color(self.board[i][j], self.color_map)
                self.canvas.create_rectangle(
                    x1,y1,x2,y2,
                    fill=cell_color,
                    outline='#34495E',
                    width=2
                )
        if queens :
            for qr, qc in queens :
                x = qc * self.cell_size + self.cell_size // 2
                y = qr * self.cell_size + self.cell_size // 2

                self.canvas.create_text(
                    x, y,
                    text="♛",
                    font=('Arial', int(self.cell_size * 0.6)),
                    fill="#000000"
                )

        self.canvas.update()

    def visualization_call(self, placed_queens: list[tuple[int,int]], iterations:
int) :
        self.draw_board(placed_queens)
        self.iteration_label.config(text=f"Iteration: {iterations}")
        self.root.update()

    def start_solving(self) :
        self.solve_button.config(state=tk.DISABLED)
        self.iteration_label.config(text="Searching for a solution ...")
        self.time_label.config(text="Time: 0 ms")

        success, solution, iterations, exec_time, self.queens =
self.solver_call(self.visualization_call)

        self.solution = solution
        self.execution_time = exec_time

        if success:
            self.draw_board(self.queens)

            self.iteration_label.config(text=f"Solution Found! Iteration:
{iterations}")
            self.time_label.config(text=f"Time: {exec_time:.2f} ms")

            self.save_image_button.config(state=tk.NORMAL)
            self.save_text_button.config(state=tk.NORMAL)

            messagebox.showinfo(
                "Success",
                f"Solution Found!\n\nIteration: {iterations}\nTime: {exec_time:.2f}
ms"
            )
        else:
            self.iteration_label.config(text=f"No Solution. Iteration:
{iterations}")
            self.time_label.config(text=f"Time: {exec_time:.2f} ms")
            messagebox.showwarning(
                "No Solution",
                f"There is no valid solution for this board.\n\nIteration:
{iterations}\nTime: {exec_time:.2f} ms"
```

```
            )

        self.solve_button.config(state=tk.NORMAL)

    def save_as_image(self) :
        if not self.queens:
            messagebox.showwarning("Warning", "There is no solution to save!")
            return

        filename = filedialog.asksaveasfilename(
            defaultextension=".png",
            filetypes=[("PNG Image", "*.png"), ("All Files", "*.*")]
        )

        if filename:
            try :
                save_image(self.board, self.queens, filename, cell_size=80)
                messagebox.showinfo("Success", f"Image saved successfully 
to:\n{filename}")
            except Exception as e:
                messagebox.showerror("Error", f"Failed to save file: {e}")

    def save_as_text(self):
        if not self.solution:
            messagebox.showwarning("Warning", "There is no solution to save!")
            return

        filename = filedialog.asksaveasfilename(
            defaultextension=".txt",
            filetypes=[("Text File", "*.txt"), ("All Files", "*.*")]
        )

        if filename:
            try:
                save_text(self.solution, filename)
                messagebox.showinfo("Success", f"Solution saved successfully 
to:\n{filename}")
            except Exception as e:
                messagebox.showerror("Error", f"Failed to save file: {e}")

    def run(self):
        self.root.mainloop()
```

3. **main.py**

Berfungsi sebagai entry point. Pengguna akan diberikan pilihan untuk menjalankan program dalam mode CLI (Terminal) atau GUI. Berikut adalah kodenya :

```
import sys
from solver import board_solver
from utils import read_file, save_text, save_image, GUI
import tkinter as tk
from tkinter import filedialog
```

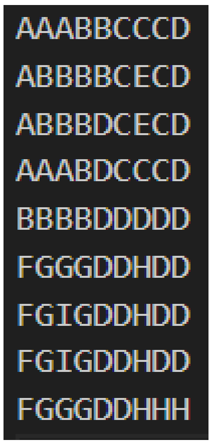
```python
# Fungsi untuk print papan
def print_board(board: list[list[str]]):
    for row in board:
        print(''.join(row))

# Fungsi untuk output berupa Command Line Interface (CLI)
def cli_mode():
    print()

    filename = input("Enter the input file name (Example: test1.txt): ").strip()

    try:
        print(f"\nRead the file : '{filename}'...")
        board = read_file(filename)
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found!")
        return
    except ValueError as e:
        print(f"Error: {e}")
        return
    except Exception as e:
        print(f"Error: {e}")
        return

    print(f"\nBoard Size: {len(board)}x{len(board)}")
    print("\nInitial Board:")
    print_board(board)
    print()

    input("Press enter to continue...")

    solver = board_solver(board)

    # Fungsi untuk visualisas CLI
    def cli_visualization(placed_queens, iterations):
        print(f"\nIteration: {iterations}")

        temp_board = [row[:] for row in board]
        for qr, qc in placed_queens:
            temp_board[qr][qc] = '#'

        for row in temp_board:
            row_str = ""
            for cell in row:
                row_str += cell + " "
            print(row_str)

        print()

    print("\nSearching for solution...\n")
    success, solution, iterations, exec_time, queens =
solver.solve(cli_visualization)
```

```python
    print("Result :")
    print()

    if success:
        print("Solution Found!")
        print()
        print_board(solution)
        print()
        print(f"Execution Time: {exec_time:.2f} ms")
        print(f"Cases Reviewed: {iterations} cases")
        print()

        save_choice = input("Save the solution as text? (y/n): ").strip().lower()
        if save_choice in ['y', 'yes', 'ya']:
            output_filename = input("Output file name (Example: solution.txt): 
").strip()
            try:
                save_text(solution, output_filename)
                print(f"Solution saved successfully to '{output_filename}'")
            except Exception as e:
                print(f"Error saving file: {e}")

        image_choice = input("\nSave the solution as a PNG image? (y/n): 
").strip().lower()
        if image_choice in ['y', 'yes', 'ya']:
            image_filename = input("Image file name (Example: solution.png): 
").strip()
            try :
                save_image(board, queens, image_filename)
                print(f"Image saved successfully to '{image_filename}'")
            except Exception as e:
                print(f"Error saving file: {e}")
    else:
        print("There is no valid solution for this board")
        print()
        print(f"Execution Time: {exec_time:.2f} ms")
        print(f"Cases Reviewed: {iterations} cases")

    print()
    print("Finish.")

# Fungsi untuk output berupa GUI
def run_gui_mode():
    root = tk.Tk()
    root.withdraw()

    filename = filedialog.askopenfilename(
        title="Select Input File",
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
    )

    if not filename:
        print("No files selected. Exit the program..")
        return

    try:
        board = read_file(filename)
    except Exception as e:
        root = tk.Tk()
        root.withdraw()
        from tkinter import messagebox
        messagebox.showerror(f"Error! Failed to read file:\n{e}")
        return

    root.destroy()

    solver = board_solver(board)

    def solve_visualization(visualization_call):
        return solver.solve(visualization_call)

    gui = GUI(board, solve_visualization)
    gui.run()

# Main Code untuk menjalankan
def main():
    print("--- Brute Force Algorithm for Queens Linkedin Problem ---")
    print()
    print("Select mode:")
    print("1. CLI Mode (Command Line Interface)")
    print("2. GUI Mode (Graphical User Interface)")
    print()

    choice = input("Your choice (1/2): ").strip()

    if choice == '1':
        cli_mode()
    elif choice == '2':
        run_gui_mode()
    else:
        print("Invalid selection!")
        sys.exit(1)


if __name__ == "__main__":
    main()
```

C. EKSPERIMEN

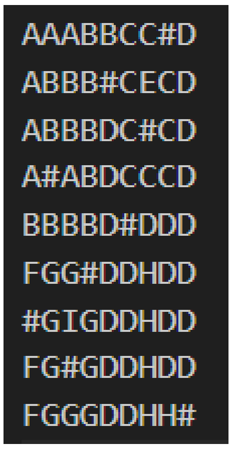
C.1 Test 1 :

Input :

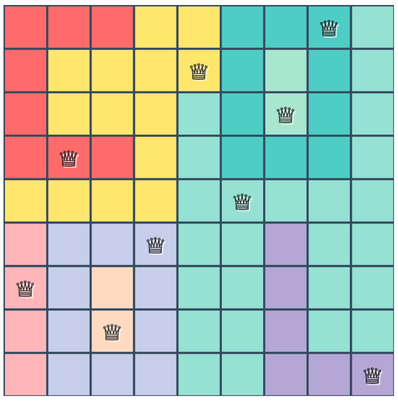


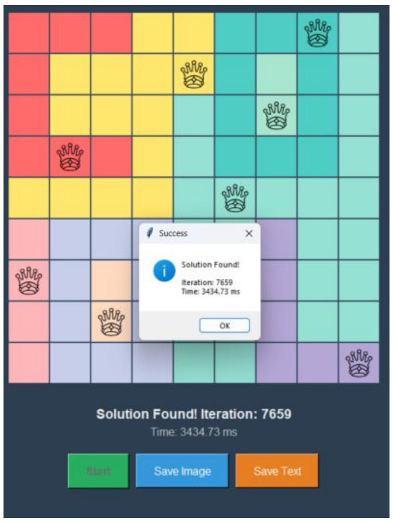
```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

Output :



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





Success

Solution Found!

Iteration: 7659
Time: 3434.73 ms

Solution Found! Iteration: 7659

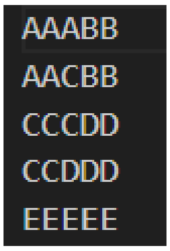
Time: 3434.73 ms

Execution Time: 423.64 ms

Cases Reviewed: 7659 cases

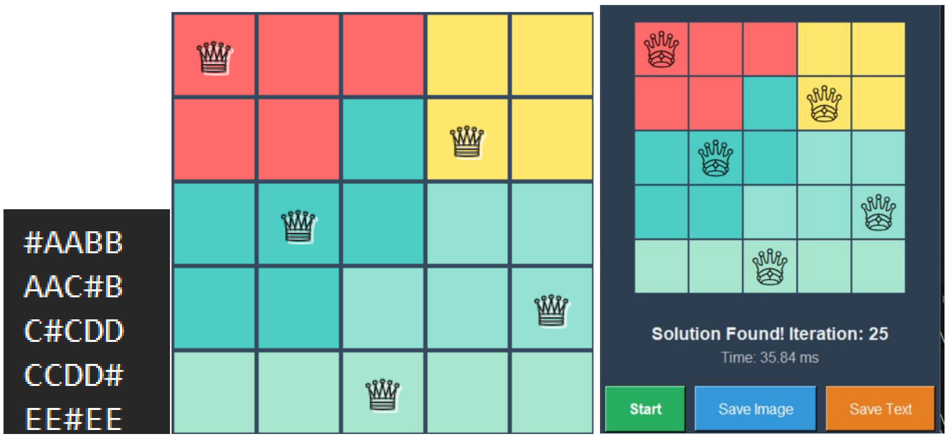
C.2 Test 2 :

Input :



```
AAABB
AACBB
CCCDD
CCDDD
EEEEE
```

Output :



```
#AABB
AAC#B
C#CDD
CCDD#
EE#EE
```

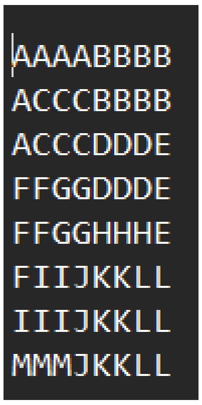
Solution Found! Iteration: 25
Time: 35.84 ms

Execution Time: 3.01 ms

Cases Reviewed: 25 cases

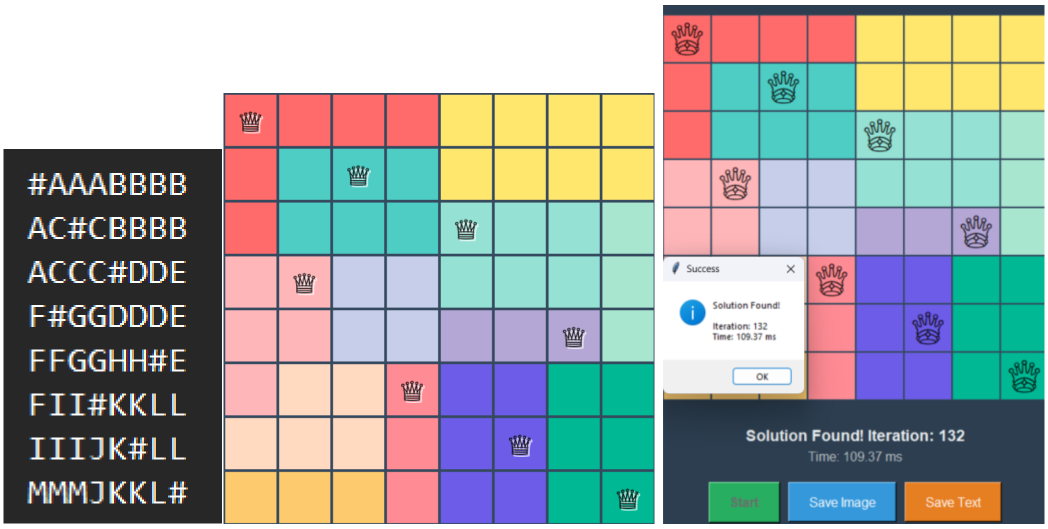
C.3 Test 3 :

Input :



```
AAAABBBB
ACCCBBBB
ACCCDDDE
FFGGDDDE
FFGGHHHE
FIIJKKLL
IIIJKKLL
MMMJKKLL
```

Output :



```
#AAABBBB
AC#CBBBB
ACCC#DDE
F#GGDDDE
FFGGHH#E
FII#KKLL
IIIJK#LL
MMMJKKL#
```

Success

Solution Found!
Iteration: 132
Time: 109.37 ms

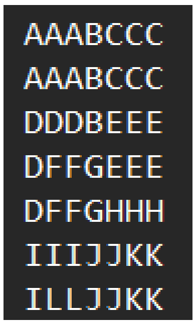
Solution Found! Iteration: 132
Time: 109.37 ms

Execution Time: 13.03 ms

Cases Reviewed: 132 cases

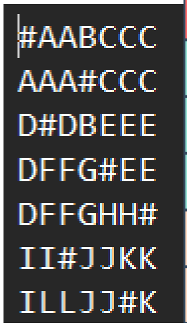
C.4 Test 4 :

Input :

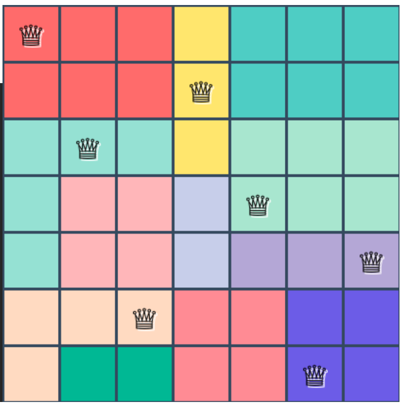


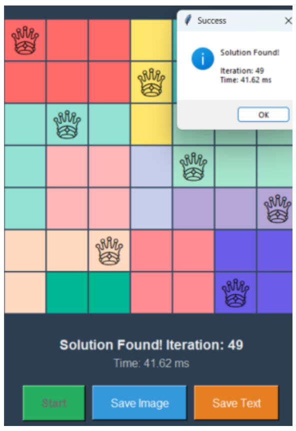
```
AAABCCC
AAABCCC
DDDBEEE
DFFGEEE
DFFGHHH
IIIJJKK
ILLJJKK
```

Output :



```
#AABCCC
AAA#CCC
D#DBEEE
DFFG#EE
DFFGHH#
II#JJKK
ILLJJ#K
```





Success

Solution Found!

Iteration: 49
Time: 41.62 ms

OK

Solution Found! Iteration: 49

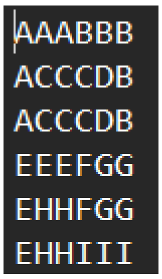
Time: 41.62 ms

Execution Time: 4.00 ms

Cases Reviewed: 49 cases

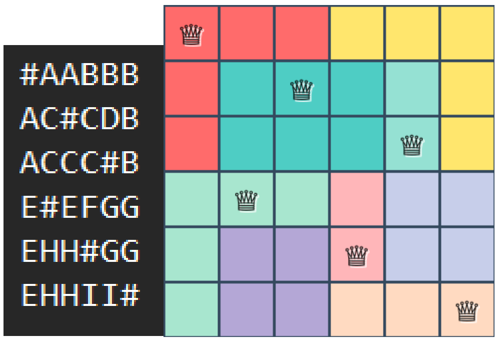
C.5 Test 5 :

Input :

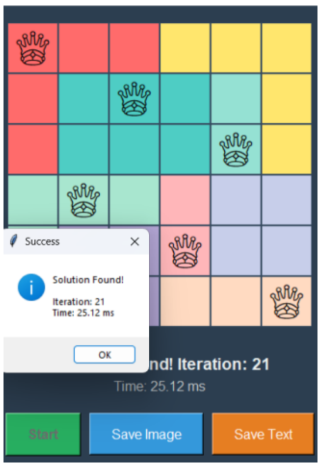


```
AAABBB
ACCCDB
ACCCDB
EEEFGG
EHHFGG
EHHIII
```

Output :



```
#AABBB
AC#CDB
ACCC#B
E#EFGG
EHH#GG
EHHII#
```



Success

Solution Found!

Iteration: 21
Time: 25.12 ms

OK

nd! Iteration: 21
Time: 25.12 ms

Execution Time: 2.14 ms

Cases Reviewed: 21 cases

LAMPIRAN

Link repository github: https://github.com/Benth48/Tucil1_10122043

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

[illegible]

Beni Lesmana

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	✓	