IF 2211 Strategi Algoritma

Penyelesaian Queens LinkedIn dengan Algoritma Brute Force

Laporan Tugas Kecil 1

Disusun untuk memenuhi tugas mata kuliah IF 2211 Strategi Algoritma pada Semester 2 Tahun Akademik 2025/2026

Disusun oleh:

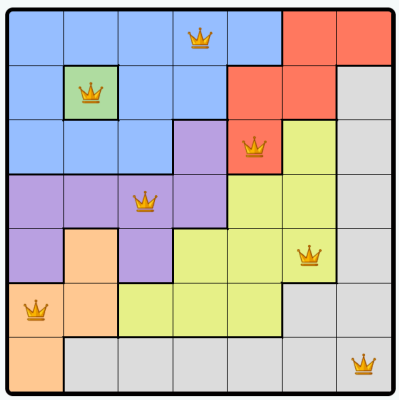
13524135 Varistha Devi

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

DESKRIPSI MASALAH



Queens adalah permainan logika yang tersedia pada situs jejaring profesional LinkedIn. Tujuan dari permainan ini adalah untuk 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, bahkan secara diagonal.

Dalam tugas kecil kali ini, penulis ditugaskan untuk membuat program sederhana dalam bahasa Python yang mengimplementasikan algoritma brute force untuk mencari solusi dalam permainan Queens, di mana algoritma brute force yang diimplementasikan tidak boleh bersifat heuristik.

Program yang dibuat harus bisa menerima input file test case berekstensi .txt yang berisi layout papan kosong dengan dimensi NxN seperti berikut:

```
AAABCD
AAABCD
BBBBCD
BBECCD
FEECCD
FFEECC
```

Kemudian, program menghasilkan output berupa solusi dari papan, waktu eksekusi, banyak kasus, dan prompt untuk menyimpan solusi dalam .txt

METODE PENYELESAIAN DAN PENJELASAN ALGORITMA

Program ini mencari solusi untuk permainan Queens dengan menggunakan metode brute force tanpa heuristik. Walau tidak ada definisi formal, brute force murni itu sendiri dapat dikatakan sebagai sebuah pendekatan problem-solving yang exhaustive dan mencoba setiap kemungkinan kombinasi atau kandidat secara sistematis. Untuk kasus ini, pendekatan brute force dilakukan dengan cara mencoba seluruh kemungkinan penempatan queen pada papan permainan dan memeriksa satu per satu apakah konfigurasi tersebut memenuhi seluruh aturan permainan. Layaknya sebuah game, tentu saja terdapat beberapa peraturan atau constraint agar suatu solusi itu dianggap valid. Dalam permainan Queens sendiri juga memiliki beberapa peraturan sederhana, yaitu:

- Setiap baris, kolom, dan wilayah berwarna harus mengandung tepat satu queen
- Queen tidak boleh ditempatkan pada sel yang saling bersebelahan, termasuk yang bersebelahan secara diagonal

Maka, untuk menyederhanakan pencarian dengan mengambil peraturan/constraint yang ada, dalam program ini digunakan representasi bahwa:

- ➔ Setiap baris hanya memiliki satu queen
- ➔ Posisi queen pada setiap baris direpresentasikan sebagai indeks kolom

Dengan demikian, sebuah solusi dapat direpresentasikan sebagai sebuah list/array berukuran N di mana indeks array menyatakan baris dan nilai array menyatakan kolom tempat queen diletakkan pada baris tersebut. Seluruh kemungkinan solusi diperoleh dengan cara melakukan permutasi secara keseluruhan dari daftar kolom [0, 1, 2, …, N-1]. Setiap permutasi merepresentasikan satu kemungkinan konfigurasi penempatan queen pada papan dan untuk tiap kombinasi dari permutasi yang dihasilkan, program lalu akan melakukan validasi berdasarkan peraturan permainan (tidak ada dua queen yang berdekatan dan setiap warna pada papan harus memiliki tepat satu queen). Proses brute force ini akan dihentikan ketika solusi valid pertama ditemukan atau ketika seluruh kemungkinan solusi telah diperiksa.

Berikut adalah langkah-langkah dari algoritma yang telah diimplementasikan:

1. Program membaca file input .txt dan membentuk papan permainan dalam bentuk matriks dua dimensi
2. Program menentukan ukuran papan N dan daftar warna unik yang terdapat pada papan
3. Program membuat sebuah list berisi angka 0 hingga N-1 yang merepresentasikan seluruh kemungkinan kolom
4. Program menghasilkan seluruh permutasi dari list kolom tersebut menggunakan fungsi rekursif dengan teknik swap. Algoritma ini bekerja dengan menentukan satu elemen sebagai posisi pertama, kemudian membentuk cabang untuk setiap pilihan yang tersedia. Setelah satu elemen dikunci pada posisi tertentu, sisa elemen akan dipermutasi secara rekursif menggunakan teknik swap. Selama proses backtracking, bagian yang belum dikunci akan terus ditukar-tukar untuk membentuk susunan baru, sedangkan bagian yang sudah tetap tidak akan diubah. Ketika seluruh posisi telah terisi, terbentuk satu permutasi lengkap yang merepresentasikan satu kandidat solusi. Proses ini berlanjut hingga semua kemungkinan susunan telah dieksplorasi, termasuk saat elemen pertama ditukar dengan elemen terakhir sebagai cabang terakhir.
5. Untuk setiap permutasi yang dihasilkan

    a. Permutasi tersebut dianggap sebagai satu konfigurasi penempatan queen
    b. Program lalu melakukan pengecekan validitas konfigurasi, memastikan tidak ada dua queen yang saling bersebelahan dan setiap warna memiliki tepat satu queen
6. Jika konfigurasi valid ditemukan, program akan menyimpan konfigurasi itu sebagai solusi dari permainan dan proses pencarian dihentikan
7. Jika seluruh konfigurasi permutasi telah diperiksa dan tidak menemukan solusi, maka program akan menyatakan bahwa tidak ada solusi yang valid
8. Program menampilkan solusi, jumlah konfigurasi yang diperiksa, serta waktu pencarian. Selama program berjalan, ia juga akan menampilkan visualisasi dari proses brute force yang sedang berlangsung untuk setiap iterasi ke-100.

Berikut adalah pseudocode dari algoritma inti dan functions bantuan yang di-convert menggunakan converter pada [Python to Pseudocode converter · GitHub](#).

Pseudocode solver.py

```
IMPORT time

from funcs IMPORT is_valid, visualize


DEFINE FUNCTION generate_permutations(array, idx=0):

    IF idx EQUALS len(array):

        yield array.copy()

        RETURN

    FOR i IN range(idx, len(array)):

        SET array[idx], array[i] TO array[i], array[idx]

        yield from generate_permutations(array, idx + 1)

        SET array[idx], array[i] TO array[i], array[idx]


SET def solve_game(board, colors, visual_each TO 100):

    SET n TO len(board)

    SET iterations TO 0

    SET start TO time.time()
```

```
    SET cols TO list(range(n))
    SET solution TO None

    FOR p IN generate_permutations(cols):
        iterations += 1
        IF iterations % visual_each EQUALS 0:
            CALL #F visualize(board, p, iterations)
        IF is_valid(p, board, colors):
            SET solution TO p
            break

    SET elapsed TO (time.time() - start) * 1000
    RETURN solution, iterations, elapsed
```

Pseudocode funcs.py

```
DEFINE FUNCTION is_valid(positions, board, colors):
    SET n TO len(positions)
    FOR i IN range(n):
        FOR j IN range(i + 1, n):
            #cek adjacency
            IF abs(i - j) <= 1 and abs(positions[i] - positions[j])
<= 1:
                RETURN False
    #cek apakah semua warna udah punya queen
    SET seen TO set()
    FOR row, col IN enumerate(positions):
```

```
        seen.add(board[row][col])
    RETURN len(seen) EQUALS len(colors)


DEFINE FUNCTION visualize(board, positions, iteration):
    OUTPUT(f"Iterasi ke-{iteration}:")
    SET temp TO [row[:] FOR row IN board]
    FOR row, col IN enumerate(positions):
        SET temp[row][col] TO '#'
    FOR row IN temp:
        OUTPUT("".join(row))


DEFINE FUNCTION OUTPUT_solution(board, positions):
    SET temp TO [row[:] FOR row IN board]
    FOR row, col IN enumerate(positions):
        SET temp[row][col] TO '#'
    FOR row IN temp:
        OUTPUT("".join(row))
```

Sebagai catatan tambahan, menurut saya meskipun program ini menerapkan beberapa constraint untuk memaksimalkan kinerja program, program ini masih layak untuk dikategorikan sebagai brute force tanpa heuristik karena:

- ➢ Program mencari seluruh kemungkinan konfigurasi/solusi tanpa menghilangkan kemungkinan di awal (tidak adanya early pruning)
- ➢ Tidak menggunakan heuristik apapun, mengingat bahwa heuristik ≠ constraint
- ➢ Validasi hanya dilakukan setelah sebuah konfigurasi lengkap terbentuk

Terkait implementasi permutasi yang dilakukan dengan menggunakan rekursi, ia hanya berfungsi sebagai mekanisme pembangkitan kemungkinan dan tidak memengaruhi ruang pencarian.

SOURCE CODE PROGRAM

Berikut adalah source code dari file-file program yang telah diimplementasikan.

- file_management.py

```
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent
TEST_DIR = BASE_DIR / "test"

def parse_file(filename):
    path = TEST_DIR / filename
    if not path.exists():
        raise FileNotFoundError("File tidak ditemukan. Pastikan file terdapat dalam folder test.")
    if path.suffix != ".txt":
        raise ValueError("Format file harus .txt")
    lines = [line.strip() for line in path.read_text().splitlines() if line.strip()]
    board = [list(line) for line in lines]
    size = len(board)
    if size == 0:
        raise ValueError("File kosong.")
    if any(len(row) != size for row in board):
        raise ValueError("File harus berdimensi N x N.")
    if size > 26:
        raise ValueError("Karena menggunakan huruf sebagai warna, maka ukuran maksimum yang
diperbolehkan adalah 26 x 26.")
    colors = sorted({char for row in board for char in row}) #ini bakalan dipake buat validasi nanti apakah
masing-masing warna udah punya queen atau belum
    if len(colors) != size:
        raise ValueError("Jumlah warna harus sama dengan ukuran papan atau jumlah baris ataupun kolom.")
    return board, colors

def save_solution_to_file(filename, board, positions):
    path = TEST_DIR / filename
    temp = [row[:] for row in board]
    for row, col in enumerate(positions):
        temp[row][col] = '#'
    with open(path, 'w') as f:
        for row in temp:
```

```
            f.write("".join(row) + "\n")
```

- ❖ funcs.py

```
def is_valid(positions, board, colors):
    n = len(positions)
    for i in range(n):
        for j in range(i + 1, n):
            #cek adjacency
            if abs(i - j) <= 1 and abs(positions[i] - positions[j]) <= 1:
                return False
    #cek apakah semua warna udah punya queen
    seen = set()
    for row, col in enumerate(positions):
        seen.add(board[row][col])
    return len(seen) == len(colors)

def visualize(board, positions, iteration):
    print(f"Iterasi ke-{iteration}:")
    temp = [row[:] for row in board]
    for row, col in enumerate(positions):
        temp[row][col] = '#'
    for row in temp:
        print("".join(row))

def print_solution(board, positions):
    temp = [row[:] for row in board]
    for row, col in enumerate(positions):
        temp[row][col] = '#'
    for row in temp:
        print("".join(row))
```

- ❖ main.py

```
from file_management import parse_file, save_solution_to_file
from solver import solve_game
from funcs import print_solution

def main():
    filename = input("Input nama file (pastikan file berada di folder test): ")
    board, colors = parse_file(filename)
    solution, iterations, elapsed = solve_game(board, colors)
```

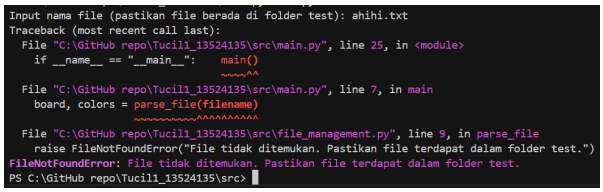
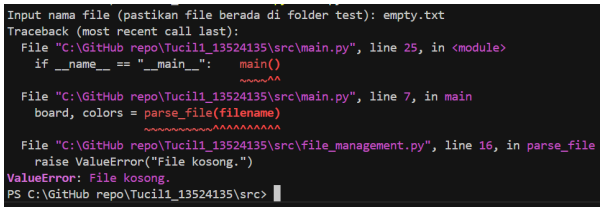
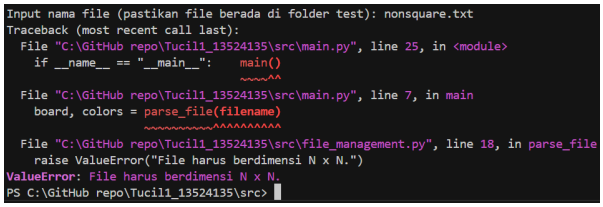
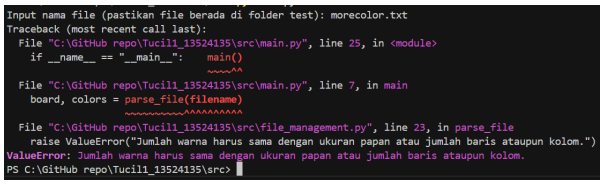
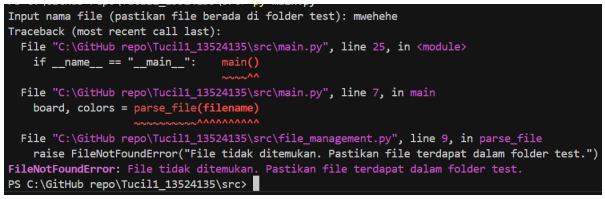
```
    if solution is not None:
        print("\nSolusi ditemukan!")
        print_solution(board, solution)
        print("\n")
        print(f"Posisi queen di setiap baris: {solution}")
        print(f"Jumlah solusi yang ditinjau: {iterations}")
        print(f"Waktu pencarian: {elapsed:.2f} ms")
        choice = input("Apakah anda ingin menyimpan solusi ke file? WARNING: akan overwrite file yang sudah ada! (y/n): ").strip().lower()
        if choice == 'y':
            save_solution_to_file(filename, board, solution)
            print(f"Solusi berhasil disimpan ke {filename}")
        else:
            print("Solusi tidak disimpan.")
    else:
        print("Tidak ada solusi yang valid ditemukan.")
        print(f"Total time elapsed: {elapsed:.2f} ms")
if __name__ == "__main__":    main()
```

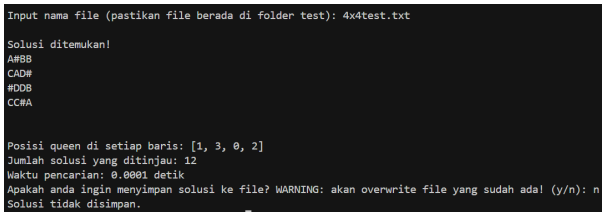
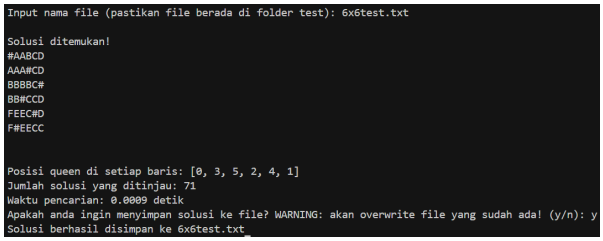
- ❖ solver.py

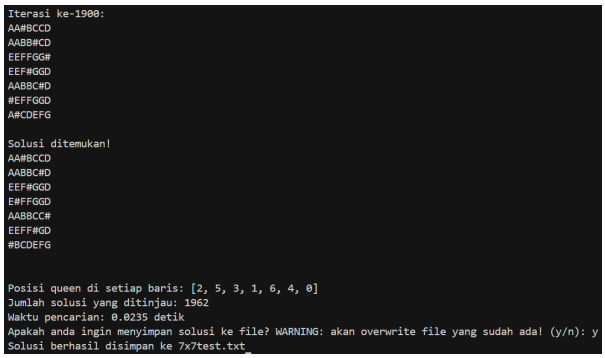
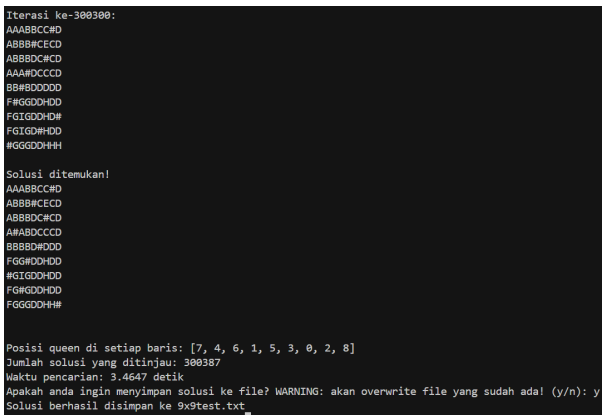
```
import time
from funcs import is_valid, visualize

# generate semua permutasi dari possible queen positions untuk tiap baris itu di kolom keberapa aja
def generate_permutations(array, idx=0):
    if idx == len(array):
        yield array.copy()
        return
    for i in range(idx, len(array)):
        array[idx], array[i] = array[i], array[idx]
        yield from generate_permutations(array, idx + 1)
        array[idx], array[i] = array[i], array[idx]
# Note: fungsi ini menggunakan rekursi dan teknik swap untuk menghasilkan seluruh permutasi. Walau secara implementasi mirip dengan backtracking ia tidak melakukan pruning sama sekali, dan juga hanya digunakan untuk generate permutasi sebagai pengganti library itertools.
# Referensinya: https://jrwalsh1.github.io/posts/permutations-and-the-n-queens-problem/

def solve_game(board, colors, visual_each = 100):
    n = len(board)
    iterations = 0
```

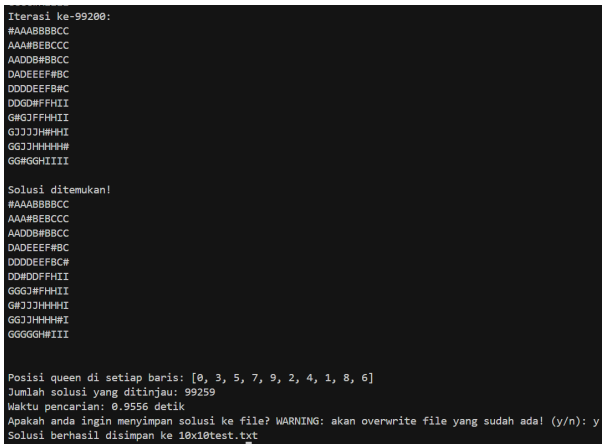
```
    start = time.time()

    cols = list(range(n))
    solution = None

    for p in generate_permutations(cols):
        iterations += 1
        if iterations % visual_each == 0:
            visualize(board, p, iterations) #disini terjadi visualisasi tiap 100 iterasi, 100 karena kalau 10 takut kebanyakan
        if is_valid(p, board, colors):
            solution = p
            break

    elapsed = (time.time() - start) * 1000 #ubah ke ms
    return solution, iterations, elapsed
# Alur dari algoritma ini
# - Generate dulu semua permutasi dari posisi queen yang mungkin untuk tiap baris (misal untuk board 3x3 itu berarti generate permutasi dari [0, 1, 2] yang berarti queen di baris pertama bisa di kolom 0, queen di baris kedua bisa di kolom 1, dan queen di baris ketiga bisa di kolom 2)
# - Tiap permutasi yang digenerate akan divalidasi menggunakan fungsi is_valid (apakah semua warna sudah ada queen, apakah ada yang adjacent, apakah ada yang bentrok secara kolom)
# - Kalau ketemu solusi yang valid, langsung berhenti dan ngasih solusinya
# - Selama program berjalan, tiap 100 iterasi akan nongol visualisasi board dengan posisi queen yang lagi diuji coba
```

PENGUJIAN PROGRAM

Input	Output	Tangkapan Layar
ahihi.txt atau file .txt yang tidak terdapat dalam folder test	File tidak ditemukan. Pastikan file terdapat dalam folder test.	
File .txt kosong	File kosong.	
File dengan isi berdimensi bukan persegi seperti AABB CADB CDDB	File harus berdimensi N x N.	
File dengan papan yang memiliki lebih banyak warna dibandingkan kolom dan baris seperti AABBCCDDEE AABBCCDDEE FFGGHHIIJJ FFGGHHIIJJ KKLLMMNNOO KKLLMMNNOO PPQQRRSSTT PPQQRRSSTT UUVVWWXXYY UUVVWWXXYY	Jumlah warna harus sama dengan ukuran papan atau jumlah baris ataupun kolom.	
mwehehe atau input asal apapun	File tidak ditemukan. Pastikan file terdapat dalam folder test.	

File valid berukuran 4x4 dan tidak simpan solusi	Solusi ditemukan! A#BB CAD# #DDB CC#A Posisi queen di setiap baris: [1, 3, 0, 2] Jumlah solusi yang ditinjau: 12 Waktu pencarian: 0.0001 detik Apakah anda ingin menyimpan solusi ke file? WARNING: akan overwrite file yang sudah ada! (y/n): n Solusi tidak disimpan.	
File valid berukuran 6x6 dan simpan solusi	Solusi ditemukan! #AABCD AAA#CD BBBBC# BB#CCD FEEC#D F#EECC Posisi queen di setiap baris: [0, 3, 5, 2, 4, 1] Jumlah solusi yang ditinjau: 71 Waktu pencarian: 0.0009 detik Apakah anda ingin menyimpan solusi ke file? WARNING: akan overwrite file yang sudah ada! (y/n): y Solusi berhasil disimpan ke 6x6test.txt	

File valid berukuran 7x7 dan simpan solusi	Visualisasi proses bruteforce per 100 iterasi Solusi ditemukan! AA#BCCD AABBC#D EEF#GGD E#FFGGD AABBCC# EEFF#GD #BCDEFG Posisi queen di setiap baris: [2, 5, 3, 1, 6, 4, 0] Jumlah solusi yang ditinjau: 1962 Waktu pencarian: 0.0235 detik Apakah anda ingin menyimpan solusi ke file? WARNING: akan overwrite file yang sudah ada! (y/n): y Solusi berhasil disimpan ke 7x7test.txt	
File valid berukuran 9x9 dan simpan solusi	Visualisasi proses bruteforce per 100 iterasi Solusi ditemukan! AAABBCC#D ABBB#CECD ABBBDC#CD A#ABDCCCD BBBBD#DDD FGG#DDHDD #GIGDDHDD FG#GDDHDD FGGGDDHH# Posisi queen di setiap baris: [7, 4, 6, 1, 5, 3, 0, 2, 8] Jumlah solusi yang ditinjau: 300387	

	Waktu pencarian: 3.4647 detik Apakah anda ingin menyimpan solusi ke file? WARNING: akan overwrite file yang sudah ada! (y/n): y Solusi berhasil disimpan ke 9x9test.txt	
File valid berukuran 10x10 dan simpan solusi	Visualisasi proses bruteforce per 100 iterasi Solusi ditemukan! #AAABBBBCC AAA#BEBCCC AADDB#BBCC DADEEEF#BC DDDDEEFBC# DD#DDFFHII GGGJ#FHHII G#JJJHHHHI GGJJHHHH#I GGGGGH#III Posisi queen di setiap baris: [0, 3, 5, 7, 9, 2, 4, 1, 8, 6] Jumlah solusi yang ditinjau: 99259 Waktu pencarian: 0.9556 detik Apakah anda ingin menyimpan solusi ke file? WARNING: akan overwrite file yang sudah ada! (y/n): y Solusi berhasil disimpan ke 10x10test.txt	

LAMPIRAN

Tabel Checklist Spesifikasi Program

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		✓

Repository Program

Repository program dapat dibuka melalui link berikut:
https://github.com/Beeziebloop/Tucil1_13524135

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

Varistha Devi