Penyelesaian Permainan *Queens* Linkedin dengan Algoritma *Brute Force*

Oleh:
Izhar Alif Akbar / 18223129

INSTITUT TEKNOLOGI BANDUNG
2026

Daftar Isi

BAB I
Pendahuluan

1.1. Deskripsi Persoalan

Permainan **Queens LinkedIn** merupakan sebuah persoalan logika yang populer pada platform LinkedIn.. Tujuan utama dari permainan ini adalah menempatkan sejumlah N ***Queen*** pada papan berukuran $N \times N$ yang terdiri dari sel-sel berwarna. Permainan ini memiliki aturan ketat yang harus dipenuhi agar solusi dianggap valid:

1. **Aturan Baris dan Kolom:** Setiap baris dan setiap kolom hanya boleh diisi oleh tepat satu Queen.
2. **Aturan Warna**: Setiap wilayah warna (*region*) yang terhubung hanya boleh memiliki tepat satu *Queen*.
3. ***Adjacency***: Dua buah *Queen* tidak boleh bersentuhan satu sama lain, baik secara horizontal, vertikal, maupun diagonal.

1.2. Batasan

Batasan pada persoalan ini merupakan kondisi input yang dianggap valid:

1. Jumlah baris sama dengan jumlah kolom
2. Jumlah warna tidak lebih dari jumlah baris atau kolom
3. Input warna pada papan direpresentasikan dengan alfabet

BAB II
Pembahasan

2.1. Algoritma Brute Force (Exhaustive Search)

Untuk menyelesaikan persoalan ini, digunakan pendekatan ***Brute Force*** jenis ***Exhaustive Search***. Algoritma ini bekerja dengan cara membangkitkan semua kemungkinan konfigurasi penempatan Queen dan menguji kevalidannya satu per satu hingga ditemukan solusi yang memenuhi seluruh aturan.

Berdasarkan strategi yang diterapkan, langkah-langkah algoritma adalah sebagai berikut:

1. **Inisialisasi**
   Program membaca matriks input yang berisi karakter warna. Dimensi papan $N \times N$ ditentukan dari jumlah baris dan kolom input.
2. **Ruang Pencarian**
   Karena aturan mengharuskan setiap baris dan kolom memiliki tepat satu Queen, maka ruang pencarian dapat direduksi menjadi persoalan kombinasi dan permutasi:
   i. **Kombinasi baris**: Setiap *Queen* akan diletakkan pada baris yang berbeda, sehingga otomatis syarat *Queen* di baris yang berbeda akan terpenuhi. Terdapat kemungkinan jumlah baris (n) lebih dari jumlah *queen* (q), sehingga total kemungkinannya dapat dihitung dengan $C_{(n,\,q)}$.
   ii. **Permutasi kolom**: Setelah masing-masing *Queen* ditempatkan dalam baris yang berbeda, tahapan selanjutnya dengan menggeser kolom *Queen* (n) di setiap baris hingga mendapatkan konfigurasi yang diinginkan, sehingga total kemungkinannya dapat dihitung dengan $P_{(n,\,q)}$.

   Maka, ruang pencarian pada kasus Queens Linkedin:

   Total Konfigurasi: $C_{(r,\,q)} * P_{(c,\,q)}$

   Kompleksitas: $O\left(\frac{(n!)^2}{q!((n-q)!)^2}\right)$

3. **Pemeriksaan Validitas**
   Untuk setiap permutasi (konfigurasi) yang didapatkan, dilakukan pemeriksaan aturan permainan:
   i. **Cek Baris dan Kolom**: Memastikan hanya terdapat satu *Queen* di setiap baris dan kolom.
   ii. **Cek Warna**: Memastikan hanya terdapat satu *Queen* di setiap wilayah warna.

iii. ***Adjacency***: Memastikan tidak ada dua *Queen* yang bersebelahan. Untuk setiap pasangan *Queen* pada (i, c_i) dan (j, c_j), jarak Euclidean atau selisih koordinat diperiksa. Jika $|i - j| \leq 1$ dan $\left|c_i - c_j\right| \leq 1$, maka konfigurasi tidak valid.

4. Terminasi

i. Jika ditemukan solusi yang valid, maka algoritma akan berhenti, menyimpan solusi, dan mencetak hasilnya.

ii. Jika seluruh kemungkinan telah dicoba dan tidak ditemukan solusi, maka program menyimpulkan bahwa tidak ada solusi.

2.2. Implementasi Kode Program

Untuk mempermudah proses implementasi, dilakukan dekomposisi menjadi fungsi utama dan beberapa fungsi pendukung.

2.2.1 Program Utama

Program utama merupakan tempat utama program dijalankan.

```
PROCEDURE SolveGenerator(board, colors)

    n ← panjang(board)
    numbers ← [0,1,2,...,n-1]

    row_combinations ← COMBINATIONS(numbers,
panjang(colors))
    column_permutations ← PERMUTATIONS(numbers,
panjang(colors))

    count ← 0

    FOR setiap rows ∈ row_combinations DO
        FOR setiap cols ∈ column_permutations DO

            positions ← pasangan elemen rows dan
cols (ZIP)

            count ← count + 1

            is_solution ← CheckConstraint(positions)

            OUTPUT (positions, count, is_solution)

            IF is_solution = TRUE THEN
                STOP
            ENDIF

        ENDFOR
    ENDFOR

END PROCEDURE
```

Pseudocode

```
   def solve_generator(self):
        n = [i for i in range(len(self.board))]

        row_combination = self._comb(n,
len(self.colors))
        column_combination = self._permute(n,
len(self.colors))

        count = 0
        for rows in row_combination:
            for cols in column_combination:
                positions = list(zip(rows,
cols))
                count += 1

                is_solution =
self.check_constraint(positions)

                yield positions, count,
is_solution

                if is_solution:
                    return
```

Python

2.2.2 Program Pendukung

Program pendukung dalam kasus ini adalah beberapa fungsi yang akan digunakan di program utama untuk menyelesaikan persoalan.

A. Fungsi Ambil Warna Unik

```
FUNCTION GetColors(board)
    s ← himpunan kosong

    FOR setiap row dalam board DO
        FOR setiap karakter c dalam row DO
            tambahkan c ke s
        ENDFOR
    ENDFOR

    RETURN list(s)
END FUNCTION
```

Pseudocode

```
   def _permute(self, arr, k):
        if k == 0:
            return [[]]
        result = []
        for i in range(len(arr)):
            remaining = arr[:i] + arr[i+1:]
            for p in self._permute(remaining,
k-1):
                result.append([arr[i]] + p)
        return result
```

Python

B. Fungsi Parser Board

```
FUNCTION ParseBoardFromTxt(file_path)
    buka file file_path sebagai f
    lines ← semua baris file tanpa newline
    RETURN lines
END FUNCTION
```

Pseudocode

```
   def parse_board_from_txt(file_path):
        with open(file_path, 'r') as f:
            lines = f.read().splitlines()
        return lines
```

Python

C. Fungsi Permutasi k Elemen

```
FUNCTION Permute(arr, k)

    IF k = 0 THEN
        RETURN [list kosong]
    ENDIF

    result ← []

    FOR i dari 0 sampai panjang(arr)-1 DO
        remaining ← arr tanpa elemen ke-i

        subperm ← Permute(remaining, k-1)

        FOR setiap p dalam subperm DO
            tambahkan [arr[i] + p] ke result
        ENDFOR
    ENDFOR

    RETURN result
END FUNCTION
```

Pseudocode

```
   def _permute(self, arr, k):
        if k == 0:
            return [[]]
        result = []
        for i in range(len(arr)):
            remaining = arr[:i] + arr[i+1:]
            for p in self._permute(remaining,
k-1):
                result.append([arr[i]] + p)
        return result
```

Python

D. Fungsi Kombinasi k Elemen

```
FUNCTION Comb(arr, k)

    result ← []
    n ← panjang(arr)

    FOR mask dari 0 sampai (2^n - 1) DO
        subset ← []

        FOR i dari 0 sampai n-1 DO
            IF bit i pada mask aktif THEN
                tambahkan arr[i] ke subset
            ENDIF
        ENDFOR

        IF panjang(subset) = k THEN
            tambahkan subset ke result
        ENDIF
    ENDFOR

    RETURN result
END FUNCTION
```

Pseudocode

```
   def _comb(self, arr, k):
        result = []
        n = len(arr)
        for mask in range(1 << n):
            subset = []
            for i in range(n):
                if mask & (1 << i):
                    subset.append(arr[i])
            if len(subset) == k:
                result.append(subset)
        return result
```

Python

E. Fungsi Cek Baris & Kolom Unik

```
FUNCTION CheckRowsCols(arr)

    rows ← []
    cols ← []

    FOR setiap (r,c) dalam arr DO
        tambahkan r ke rows
        tambahkan c ke cols
    ENDFOR

    IF panjang(rows) = jumlah unik rows
       DAN panjang(cols) = jumlah unik cols
    THEN
        RETURN TRUE
    ELSE
        RETURN FALSE
END FUNCTION
```

Pseudocode

```
   def _check_rows_cols(self, arr):
        rows = []
        cols = []
        for r, c in arr:
            rows.append(r)
            cols.append(c)
        if (len(rows) == len(set(rows))) and
(len(cols) == len(set(cols))):
            return True
        return False
```

Python

F. Fungsi Cek Tidak Bersebelahan (*Adjacency*)

```
FUNCTION CheckDistance(arr)

    FOR i dari 0 sampai panjang(arr)-1 DO
        (r1,c1) ← arr[i]

        FOR j dari i+1 sampai akhir DO
            (r2,c2) ← arr[j]
```

```
   def _check_distance(self, arr):
        for i in range(len(arr)):
            r1, c1 = arr[i]
            for j in range(i+1, len(arr)):
                r2, c2 = arr[j]
                if abs(r1-r2) <= 1 and
abs(c1-c2) <= 1:
```

`` IF |r1-r2| ≤ 1 DAN |c1-c2| ≤ 1 THEN RETURN FALSE ENDIF ENDFOR ENDFOR RETURN TRUE END FUNCTION ```	``` return False return True ```
Pseudocode	Python

G. Fungsi Cek Warna Tidak Duplikat

`` FUNCTION CheckColors(arr, board) seen ← [] FOR setiap (r,c) dalam arr DO color ← board[r][c] IF color ada di seen THEN RETURN FALSE ENDIF tambahkan color ke seen ENDFOR RETURN TRUE END FUNCTION ```	``` def _check_colors(self, arr): seen = [] for r, c in arr: color = self.board[r][c] if color in seen: return False seen.append(color) return True ```
Pseudocode	Python

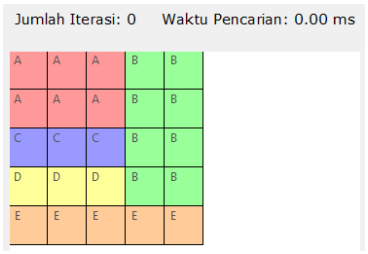
G. Fungsi Cek Semua *Constraint*

`` FUNCTION CheckConstraint(arr, board) RETURN CheckRowsCols(arr) AND CheckDistance(arr) AND CheckColors(arr, board) END FUNCTION ```	``` def check_constraint(self, arr): return (self._check_rows_cols(arr) and self._check_distance(arr) and self._check_colors(arr)) ```
Pseudocode	Python

2.3. Eksperimen dan Hasil

Berikut adalah tangkapan layar dari hasil eksekusi program untuk 5 contoh kasus uji yang berbeda. Tangkapan layar mencakup input papan, visualisasi output solusi, waktu eksekusi, dan jumlah iterasi.

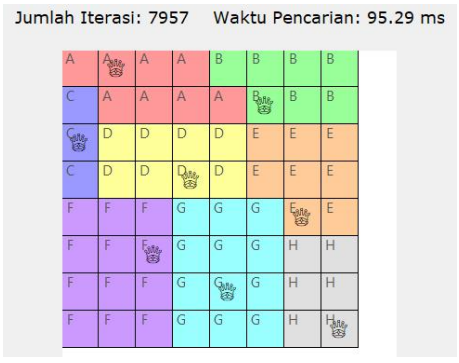
1. Test Case 1

	
Input	Output

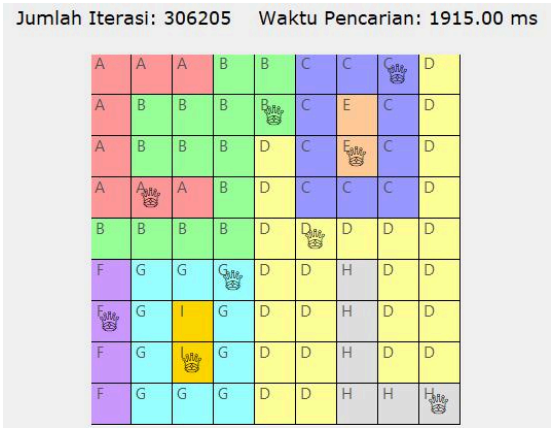
2. Test Case 2

	
Input	Output

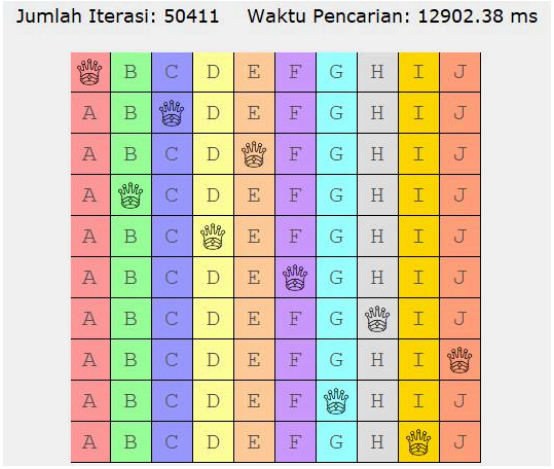
3. Test Case 3

	
Input	Output

4. Test Case 4

	
Input	Output

5. Test Case 5

	
Input	Output

Berdasarkan hasil eksperimen didapatkan hasil sebagai berikut:

Tabel 2.3. Hasil Eksperimen

Case	n	q	Iterasi	Waktu (ms)
1	5	5	37	0.41
2	7	7	339	9.00

3	8	8	7957	95.29
4	9	9	306205	1915.00
5	10	10	50411	12902.38

Berdasarkan hasil eksperimen pada Tabel 2.3, terlihat bahwa waktu eksekusi meningkat secara signifikan seiring bertambahnya ukuran papan (n). Kenaikan ini tidak bersifat linear, melainkan cenderung sangat cepat karena ruang pencarian algoritma bertumbuh secara faktorial, yaitu sebesar $C(n,\ q)\ \times\ P(n,\ q)$. Hal ini menunjukkan bahwa semakin besar nilai n, semakin banyak kemungkinan konfigurasi yang harus dibangkitkan oleh algoritma.

Hal yang menarik terlihat pada kasus ke-5, di mana jumlah iterasi lebih sedikit dibandingkan kasus ke-4, namun waktu eksekusinya lebih lama. Fenomena ini terjadi karena biaya komputasi terbesar berasal dari proses pembangkitan seluruh kombinasi baris dan permutasi kolom sebelum proses validasi dilakukan. Pada ukuran input yang lebih besar, jumlah kandidat konfigurasi meningkat sangat tajam sehingga waktu yang dibutuhkan untuk membangkitkan ruang pencarian menjadi dominan, meskipun solusi ditemukan lebih cepat.

BAB III
Kesimpulan

Berdasarkan hasil implementasi dan pengujian, dapat disimpulkan bahwa:

1. Algoritma *brute force* (*exhaustive search*) berhasil membangkitkan dan menguji seluruh kemungkinan konfigurasi penempatan *Queen* pada papan permainan.

2. Program mampu menangani aturan-aturan spesifik seperti batasan satu *Queen* per warna, satu *Queen* per baris, satu *Queen* per kolom, dan tidak bersebelahan.

3. Hasil eksperimen menunjukkan bahwa waktu eksekusi meningkat sangat cepat seiring bertambahnya ukuran papan. Hal ini disebabkan oleh pertumbuhan ruang pencarian yang bersifat faktorial, sehingga metode ini efektif untuk ukuran papan kecil hingga menengah, namun menjadi tidak efisien untuk ukuran papan besar.

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

Izhar Alif Akbar

Lampiran

Pranala Repositori

Kode program dapat diakses melalui tautan: https://github.com/Izhrr/Tucil1_18223129

Tabel Checklist

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	✓	