Laporan Tugas Kecil 1 IF2211 Strategi Algoritma

Queens LinkedIn Solver

Reinhard Alfonzo Hutabarat – 13524056
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
Email: reinhardalfonso@gmail.com, 13524056@std.stei.itb.ac.id

CONTENTS

I. Deskripsi Permasalahan

Queens adalah sebuah permainan teka-teki logika yang dipopulerkan oleh platform LinkedIn. Permainan ini dimainkan di atas sebuah papan berukuran $N \times N$ kotak. Papan tersebut bukan sekadar berupa kotak kosong, melainkan terbagi menjadi beberapa wilayah atau blok yang direpresentasikan dengan warna-warna yang berbeda. Jumlah wilayah warna pada papan selalu sama dengan dimensi papan tersebut yaitu N wilayah.

Tujuan utama dari permainan ini adalah menempatkan tepat N buah Mahkota Ratu di atas papan sedemikian rupa sehingga memenuhi seluruh aturan keseimbangan berikut:

1) Syarat Baris: Hanya boleh ada tepat satu Ratu di setiap baris.
2) Syarat Kolom: Hanya boleh ada tepat satu Ratu di setiap kolom.
3) Syarat Wilayah Warna: Hanya boleh ada tepat satu Ratu di dalam setiap blok warna yang sama.
4) Syarat Ketetanggaan: Dua buah Ratu tidak boleh diletakkan saling bersentuhan satu sama lain, baik secara vertikal, horizontal, maupun secara diagonal.

II. Pendekatan Solusi

A. Algoritma Brute Force

Algoritma Brute Force merupakan strategi pencarian sederhana yang menyelesaikan persoalan secara *straigtforward*. Algoritma ini umumnya tidak cerdas atau tidak sangkil karena ia membutuhkan biaya komputasi yang besar dan lama dalam penyelesaiannya. Meskipun begitu, metode brute force menjamin menemukan solusi persoalan, jika ada. Strategi ini tidak menggunakan optimasi atau pendekatan heuristik. Mereka bergantung pada pengujian setiap hasil potensial tanpa mengesampingkan yang lain menggunakan pemangkasan cerdas atau heuristik.

B. Memecahkan Persoalan Queens Menggunakan Algoritma Brute Force

Penyelesaian teka-teki *Queens* pada program ini diimplementasikan menggunakan algoritma *Pure Brute Force* dengan pendekatan *Generate and Test*. Program akan mencoba seluruh kemungkinan penempatan Ratu secara menyeluruh tanpa melakukan pemotongan cabang, lalu mengujinya di akhir untuk menemukan solusi yang valid. Secara konseptual, langkah-langkah pencarian ini dibagi menjadi tiga fase utama:

1) Representasi Papan
   Untuk mengoptimalkan pencarian, papan tidak dipandang sebagai matriks dua dimensi. Mengingat aturan permainan mewajibkan hanya ada satu Ratu di setiap baris, ruang pencarian disederhanakan dengan memetakan letak Ratu ke sebuah *array* satu dimensi. Pada representasi ini, indeks *array* diartikan sebagai posisi baris, sedangkan nilai yang disimpan pada indeks tersebut menunjukkan posisi kolom tempat Ratu diletakkan.
2) Fase Pembentukan
   Fase ini bertugas menyusun kombinasi letak Ratu dari baris pertama hingga baris terakhir menggunakan metode rekursif:
   - Mulai dari baris pertama, letakkan Ratu di kolom pertama.
   - Pindah ke baris berikutnya, letakkan Ratu di kolom pertama lagi.
   - Jika pengisian baris selanjutnya telah selesai dieksplorasi atau sudah mentok, algoritma akan kembali ke baris sebelumnya dan menggeser letak Ratu ke kolom berikutnya di baris yang sama.
   - Proses ini diulang terus-menerus hingga seluruh permutasi kolom di semua baris berhasil dibangkitkan.
3) Fase pengujian

Pengujian validitas suatu posisi hanya dilakukan ketika seluruh Ratu telah diletakkan di atas papan. Program kemudian memvalidasi kombinasi tersebut terhadap tiga aturan:

- Syarat Kolom: Mengecek apakah ada dua Ratu yang berada di kolom yang sama.
- Syarat Ketetanggaan: Mengecek apakah ada Ratu yang saling bersentuhan, baik di sisi atas bawah maupun diagonal.
- Syarat Warna: Membaca warna wilayah tempat setiap Ratu berpijak. Jika ada dua Ratu yang berdiri di atas warna yang sama, kombinasi digugurkan.

Jika sebuah kombinasi lolos ketiga pengujian di atas, program akan langsung berhenti dan mengembalikan kombinasi tersebut sebagai solusi akhir.

III. Source Code Program

Implementasi algoritma brute force pada program ini dibagi menjadi dua kelas utama yaitu kelas **Board** sebagai penyimpan keadaan papan dan aturan, serta kelas **Solver** sebagai mesin pencari solusi. Berikut adalah rincian dari fungsi-fungsi penting penyusun algoritmanya:

A. *Board.java*

Kelas ini bertugas menyimpan wujud fisik dari papan permainan dan memiliki fungsi untuk mengevaluasi apakah susunan Ratu di atas papan tersebut sah atau melanggar aturan.

- Representasi Keadaan
  Untuk menyimpan posisi Ratu, program tidak menggunakan matriks 2 dimensi tetapi program cukup menggunakan *array* 1 dimensi bernama queenPos.

```
public class Board {
    int n;
    char[][] boardMap;
    int[] queenPos;
}
```

  Pada representasi di atas, atribut n digunakan untuk menyimpan ukuran dari papan permainan, lalu *array* boardMap yang berbentuk 2D bertugas untuk menyimpan peta warna asli dari file input. Indeks dari *array* queenPos mewakili baris, sedangkan nilai yang disimpan di dalamnya mewakili kolom tempat Ratu diletakkan.
- Fungsi Evaluasi
  Dalam mengevaluasi atau memvalidasi aturan permainan, fungsi isBoardValid() bertugas untuk memindai susunan Ratu secara menyeluruh dan memastikan tidak ada satupun aturan permainan yang dilanggar.

```
public boolean isBoardValid() {
    boolean[] usedCol = new boolean[n];
    boolean[] usedColor = new boolean[256];

    for (int row = 0;row < n;row++) {
        int col = queenPos[row];

        if (usedCol[col]) {
            return false;
        }
        usedCol[col] = true;

        char color = boardMap[row][col];
        if (usedColor[color]) {
            return false;
        }
        usedColor[color] = true;

        if (row > 0) {
```

```
            int colQueenTop = queenPos[row-1];
            if (Math.abs(col - colQueenTop) <= 1) {
                return false;
            }
        }
    }

    return true;
}
```

Secara berurutan, fungsi di atas bekerja dengan langkah-langkah sebagai berikut:

1) Inisialisasi Memori: Program menyiapkan dua *array* boolean kosong, yaitu usedCol untuk mencatat kolom mana saja yang sudah terisi oleh Ratu, dan usedColor untuk mencatat warna apa saja yang sudah sudah ditempati.
2) Pemindaian Baris: Program melakukan perulangan dari baris pertama hingga baris terakhir untuk mengevaluasi posisi Ratu satu per satu.
3) Pengecekan Syarat Kolom: Program membaca nilai kolom dari Ratu saat ini dan melihat ke dalam atribut usedCol. Apabila nilainya sudah bernilai *true*, artinya ada Ratu lain di atasnya yang berada di kolom yang sama sehingga susunan langsung digagalkan. Jika belum, program akan menandai kolom tersebut menjadi *true*.
4) Pengecekan Syarat Warna: Program membaca matriks boardMap untuk mengetahui karakter warna tempat Ratu berpijak. Sama dengan pengecekan sebelumnya, jika warna tersebut sudah ditandai di dalam usedColor maka susunan akan digagalkan.
5) Pengecekan Syarat Ketetanggaan: Mulai dari baris kedua, program membandingkan posisi Ratu saat ini dengan posisi kolom yang berada tepat di baris atasnya. Apabila absolut jarak kedua kolom tersebut bernilai 0 (sejajar vertikal) atau 1 (bersentuhan menyilang/diagonal), susunan digagalkan.
6) Jika perulangan selesai dan tidak ada pelanggaran yang terdeteksi, program mengembalikan nilai *true* yang menandakan bahwa susunan tersebut adalah solusi yang valid.

B. `Solver.java`

Kelas ini bertugas mengeksekusi algoritma brute force murni untuk mencoba segala kemungkinan letak Ratu. Logika utamanya terletak pada fungsi rekursif solve().

```
public boolean solve(Board board, int currRow) {
    long countCase = 0;

    if (currRow == board.n) {
        countCase++;
        if (board.isBoardValid()) {
            return true;
        }
        return false;
    }

    for (int col = 0;col < board.n;col++) {
        board.queenPos[currRow] = col;
        boolean isSolutionFound = solve(board, currRow+1);

        if (isSolutionFound) {
            return true;
        }
    }
    return false;
}
```

Secara berurutan, algoritma pembentuk pada fungsi tersebut dijalankan dengan alur sebagai berikut:

1) Fase Pengujian: Pada awal fungsi, program memeriksa apakah baris saat ini sudah sama dengan dengan jumlah baris n. Kondisi ini menandakan bahwa Ratu telah diisi penuh hingga baris terakhir. Jika kondisi ini tercapai, program akan mencatat peninjauan satu kasus baru dan memanggil fungsi isBoardValid(). Jika valid, program akan mengembailkan nilai *true*. Jika salah, program mengembalikan nilai *false* agar pencarian kombinasi dilanjutkan.
2) Fase Pembangkitan Kombinasi: Apabila papan belum penuh, program masuk ke dalam blok perulangan untuk mencoba menempatkan Ratu mulai dari kolom paling kiri hingga kolom paling kanan.
3) Fase Pemanggilan Rekursif: Setelah itu Ratu diletakkan pada sebuah kolom di baris saat ini, program langsung memerintahkan dirinya sendiri secara rekursif untuk turun dan mengisi Ratu di bawahnya. Program sama sekali tidak mengecek ketepatan posisi saat masih berada di tenga-tengah perulangan ini.
4) Fase Terminasi: Jika rekursi dari baris bawahnya berhasil menemukan solusi akhir, fungsi akan segera menghentikan perulangan dan meneruskan nilai *true* tersebut ke atas. Namun jika ternyata gagal, program sekadar melanjutkan perulangan untuk menggeser letak Ratu di baris tersebut ke kolom sebelahnya.

IV. Analisis Kompleksitas

Pendekatan Brute Force yang diimplementasikan berfokus pada eksplorasi secara *exhaustive search*. Analisis kompleksitas waktunya dapat diuraikan sebagai berikut:

1) Kompleksitas Fungsi Pembentuk
   Papan permainan memiliki ukuran $N \times N$, yang berarti terdapat N baris yang masing-masing harus diisi oleh satu Ratu. Pada setiap baris, algoritma mencoba meletakkan Ratu di N kemungkinan kolom yang ada. Dengan dilakukannya proses ini secara rekursif untuk setiap baris, maka total kombinasi susunan papan yang akan dibangkitkan adalah N^N kombinasi.
2) Kompleksitas Fungsi Evaluasi
   Fungsi Evaluasi hanya dipanggil ketika N Ratu telah diletakkan. Fungsi ini melakukan perulangan sebanyak N kali untuk memeriksa setiap Ratu. Di dalam perulangan tersebut, operasi yang dilakukan hanyalah pengecekan ke dalam *array* boolean dan operasi matematika dasar, yang masing-masing membutuhkan waktu konstan. Oleh sebab itu, kompleksitas waktu untuk satu kali pemanggilan fungsi evaluasi adalah $O(N)$.

Berdasarkan kedua hal tersebut, dapat disimpulkan skenario terburuk dan terbaiknya:

- Kasus Terburuk
  Kasus terburuk terjadi ketika susunan solusi yang benar kebetulan berada pada kombinasi yang paling terakhir dihasilkan oleh fungsi pencari, atau ketika peta masukan tersebut memang sama sekali tidak memiliki solusi. Pada kondisi ini, algoritma terpaksa membangkitkan dan menguji seluruh N^N kombinasi tanpa terkecuali. Akibat dari setiap pengujian membutuhkan waktu $O(N)$, maka kompleksitas waktu pada kasus terburuk adalah hasil dari keduanya, yaitu $O(N \cdot N^N)$ atau $O(N^{N+1})$.
- Kasus Terbaik
  Kasus Terbaik terjadi apabila susunan yang benar kebetulan berada pada kombinasi pertama yang dibentuk oleh fungsi pencari. Pada kondisi ini, algoritma hanya perlu melakukan rekursif sebanyak satu lintasan ke bawah dan langsung memanggil fungsi evaluasi satu kali. Karena saat evaluasi pertama tersebut program langsung mendapatkan nilai *true*, pencarian dihentikan secara total. Oleh sebab itu, kompleksitas waktu pada kasus terbaik adalah linier, yaitu $\Omega(N)$.

V. HASIL PENGUJIAN

Berikut adalah pengujian program menggunakan berbagai konfigurasi papan masukan beserta hasil visualisasinya.

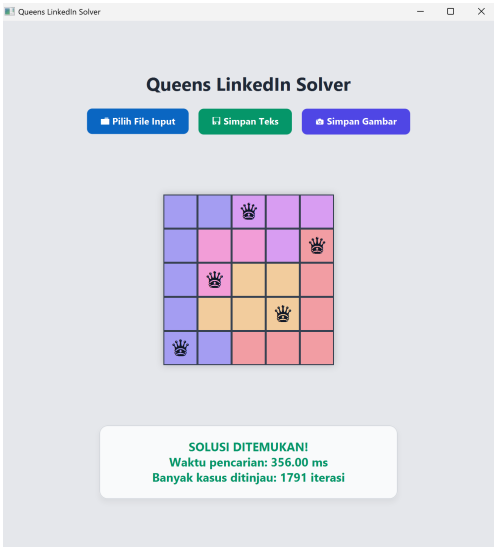
Test Case 1	
Input	**Output**
AABBB ACCBD ACEED AEEED AADDD	

Fig. 1. Hasil Uji Coba Test Case 1

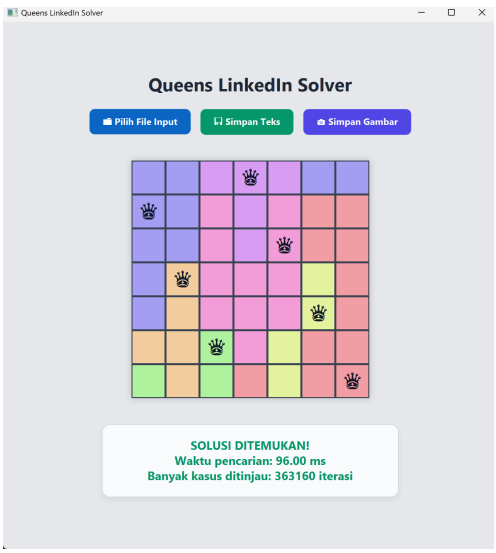
Test Case 2	
Input	**Output**
AABBBAA AACBCDD AACBCDD AECCCFD AECCCFD EEGCFDD GEGDFDD	

Fig. 2. Hasil Uji Coba Test Case 2

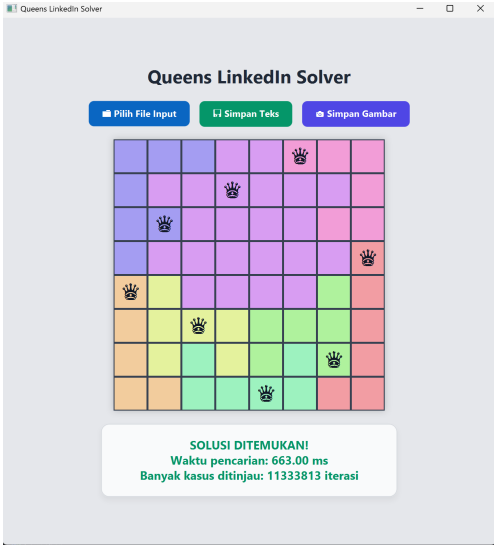
Test Case 3	
Input	**Output**
AAABBCCC ABBBBBBC AABBBBCC ABBBBBBD EFBBBBGD EFFFGGGD EFHFGHGD EEHHHHDD	

Fig. 3. Hasil Uji Coba Test Case 3

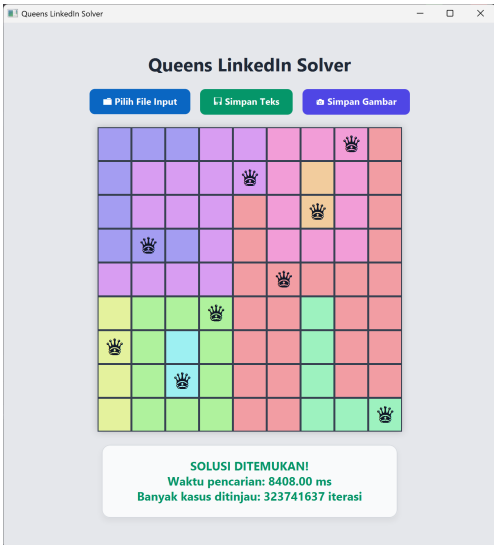
Test Case 4	
Input	**Output**
AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH	

Fig. 4. Hasil Uji Coba Test Case 4

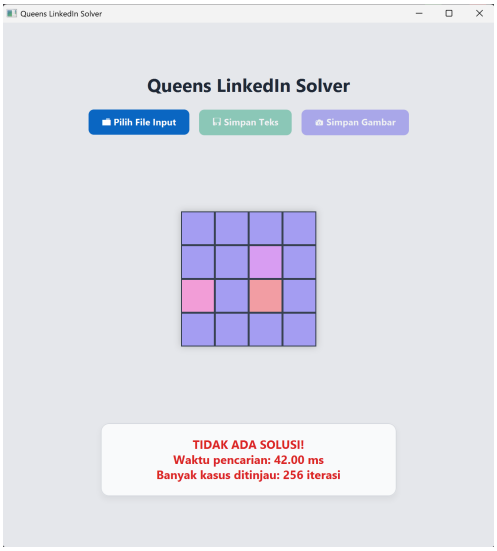
Test Case 5	
Input	**Output**
AAAA AABA CADA AAAA	

Fig. 5. Hasil Uji Coba Test Case 5

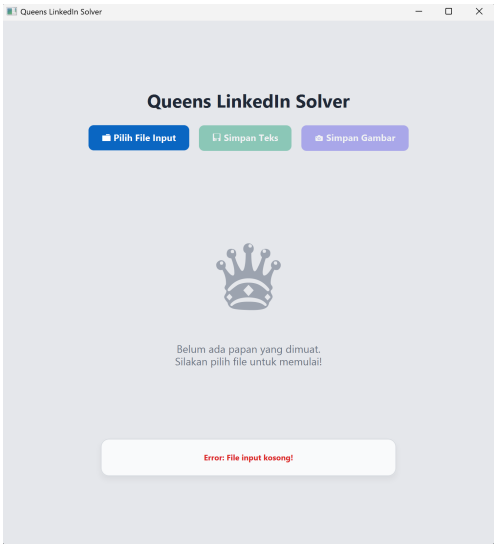
Test Case 6	
Input	**Output**
	

Fig. 6. Hasil Uji Coba Test Case 6

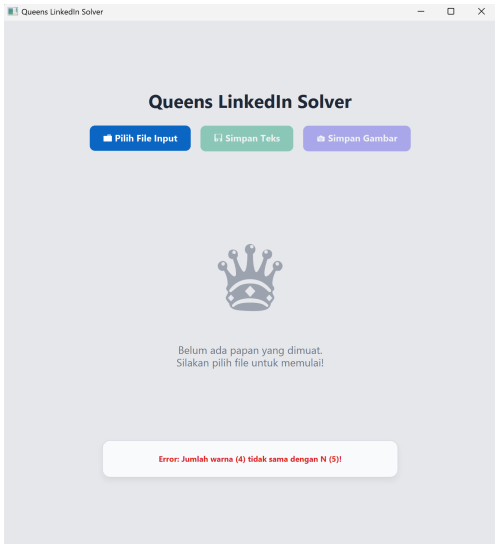
Test Case 7	
Input	**Output**
AAAAA BBBBB CCCCC DDDDD DDDDD	

Fig. 7. Hasil Uji Coba Test Case 7

VI. Pranala Repository

Seluruh kode sumber program dan *test case* dapat diakses secara penuh melalui *repository* GitHub pada tautan berikut:
https://github.com/Rizelbit/Tucil1_13524056

Pernyataan

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

Reinhard Alfonzo Hutabarat
13524056

LAMPIRAN

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	✓	