Laporan Tugas Kecil 1 IF2211 Strategi Algoritma Penyelesaian Permainan Queens LinkedIn dengan Algoritma Brute Force

Ega Luthfi Rais
13524115

Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung 2026

Bab 1. Algoritma Brute Force (Pendekatan Iteratif Odometer)

Sesuai dengan spesifikasi tugas, algoritma yang diimplementasikan untuk menyelesaikan permainan Queens ini adalah **Brute Force Murni** (*Exhaustive Search*).

Alih-alih menggunakan pendekatan rekursif (*backtracking* standar) yang membebani *stack memory*, program ini menggunakan pendekatan **Iteratif Odometer** (*Base-N Counting*). Papan permainan dipandang sebagai sebuah sistem bilangan basis-N, di mana posisi Ratu pada setiap baris merepresentasikan satu digit angka. Algoritma bekerja dengan cara membangkitkan dan menguji (*Generate & Test*) setiap kemungkinan konfigurasi dari `00...0` hingga `(N-1)(N-1)...(N-1)`.

Berikut adalah langkah-langkah rinci algoritma tersebut:

1. Inisialisasi State (Reset Odometer) Langkah pertama adalah menempatkan seluruh N buah Ratu pada kolom indeks ke-0 di setiap baris.

- **Representasi State:** Sebuah array 1D berukuran N, di mana `state[i]` menyimpan indeks kolom Ratu pada baris ke-`i`.
- **Konfigurasi Awal:** `[0, 0, 0, ..., 0]`.

2. Loop Pencarian (Exhaustion Loop) Algoritma memasuki perulangan tak terbatas (*infinite loop*) untuk memeriksa setiap kemungkinan konfigurasi satu per satu. Loop ini menjamin bahwa tidak ada satupun kemungkinan kombinasi penempatan Ratu yang terlewat (Completeness).

3. Validasi Konfigurasi (Generate & Test) Pada setiap iterasi, konfigurasi papan saat ini diperiksa validitasnya secara menyeluruh (*Rigorous Validation*). Sebuah konfigurasi dinyatakan sebagai **Solusi** jika dan hanya jika memenuhi seluruh syarat berikut:

- **Constraint Kolom:** Tidak ada dua Ratu yang menempati kolom yang sama.
- **Constraint Wilayah Warna:** Setiap wilayah warna (*color region*) yang didefinisikan pada input peta hanya boleh ditempati oleh tepat satu Ratu.
- **Constraint "No-Touching":** Tidak ada dua Ratu yang bersentuhan, baik secara vertikal, horizontal, maupun diagonal. Secara matematis, dua Ratu pada baris r1,r2 dan kolom c1,c2 bersentuhan jika $|r1-r2| \leq 1$ dan $|c1-c2| \leq 1$.

Jika konfigurasi saat ini memenuhi semua syarat di atas, algoritma mencetak papan solusi dan berhenti (terminasi sukses).

4. Mekanisme Increment (Geser Posisi) Jika konfigurasi saat ini **tidak valid**, algoritma melakukan pergeseran posisi Ratu layaknya mekanisme angka pada odometer kendaraan:

- Algoritma mencoba menggeser Ratu pada baris paling bawah (N−1) ke kanan (+1 kolom).
- **Kondisi Carry-Over:** Jika Ratu pada baris tersebut sudah berada di batas paling kanan (kolom N−1), maka Ratu tersebut dikembalikan ke kolom 0, dan algoritma mencoba menggeser Ratu pada baris di atasnya (N−2).
- Proses *carry-over* ini berlanjut terus ke atas (baris N−3, N−4, dst.) selama baris yang ditinjau sudah penuh/mentok di kanan.

5. Terminasi Gagal (No Solution) Algoritma akan berhenti dan menyatakan "Tidak Ada Solusi" jika sistem mencoba melakukan *carry-over* pada baris pertama (baris indeks 0) yang sudah berada di posisi maksimalnya (N−1). Ini menandakan bahwa seluruh ruang pencarian sebesar NN telah ditelusuri dan tidak ditemukan konfigurasi yang valid.

Bab 2. Implementasi dan Spesifikasi Program

Program dikembangkan menggunakan bahasa **C++ (Standard C++17)** dengan pendekatan berorientasi objek (*Object-Oriented Programming*). Struktur kode dipecah menjadi tiga modul utama untuk menjaga modularitas: `Parser`, `Board`, dan `Solver`.

2.1. Spesifikasi Kelas dan Fungsi

Berikut adalah spesifikasi dari kelas-kelas utama yang digunakan:

A. Class `Board` (Manajemen State Papan) Kelas ini bertanggung jawab menyimpan konfigurasi papan dan memvalidasi aturan permainan.

- **Atribut:**
  - `vector<int> queens`: Array 1D berukuran N untuk menyimpan indeks kolom Ratu pada setiap baris.
  - `vector<string> colorMap`: Matriks karakter N×N yang merepresentasikan wilayah warna.
- **Fungsi Utama:**
  - `void placeQueen(int row, int col)`: Menempatkan Ratu pada posisi (row,col).
  - `bool isValidSolution()`: Mengembalikan `true` jika konfigurasi papan saat ini valid (memenuhi seluruh constraint).
  - `void printBoard(bool clear)`: Menampilkan visualisasi papan ke terminal dengan pewarnaan ANSI.

B. Class `Solver` (Mesin Algoritma) Kelas eksekutor yang menjalankan algoritma Brute Force.

- **Fungsi Utama:**
  - `bool solve()`: Fungsi inti yang menjalankan loop iteratif (*Odometer*) untuk mencari solusi. Mengembalikan `true` jika solusi ditemukan.
  - `long long getIterations()`: Mengembalikan jumlah iterasi total yang telah dilakukan.
  - `double getTime()`: Mengembalikan waktu eksekusi dalam milidetik.

C. Class `Parser` (Input/Output Handler) Kelas utilitas statis untuk menangani operasi file.

- **Fungsi Utama:**
  - `static vector<string> readInput(string filename)`: Membaca file teks dan mengembalikannya sebagai vector string.
  - `static bool validateInput(vector<string> map)`: Memastikan input valid (persegi, karakter A-Z, jumlah warna konsisten).
  - `static void saveSolution(string filename, string content)`: Menyimpan solusi ke file eksternal.

2.2. Implementasi Kode Utama

Berikut adalah implementasi dari logika validasi (`isValidSolution`) dan algoritma pencarian iteratif (`solve`) yang menjadi inti program.

A. Validasi Constraint (`Board.cpp`) Menggunakan aritmatika selisih mutlak untuk mendeteksi pelanggaran aturan *adjacency* tanpa percabangan kompleks.

C++

```
bool Board::isValidSolution() const {
    vector<bool> colSeen(N, false);
    vector<bool> colorSeen(26, false);

    for (int r = 0; r < N; r++) {
        int c = queens[r];

        // Cek kolom unik
        if (colSeen[c]) return false;
        colSeen[c] = true;
```

```
        // Cek warna unik
        int colorIdx = colorMap[r][c] - 'A';
        if (colorSeen[colorIdx]) return false;
        colorSeen[colorIdx] = true;

        // Cek aturan "No-Touching" (vertikal & diagonal)
        if (r > 0) {
            int prevC = queens[r-1];
            if (abs(prevC - c) <= 1) return false;
        }
    }
    return true;
}
```

B. Algoritma Iteratif Odometer (`Solver.cpp`) Menggunakan `while-loop` untuk enumerasi state dari `00...0` hingga `NN...N` guna menghindari *stack overhead*.

C++

```
bool Solver::solve() {
    // Inisialisasi posisi awal (0,0,...,0)
    for (int i = 0; i < N; i++) board.placeQueen(i, 0);

    while (true) {
        iterations++;

        // Cek validitas konfigurasi saat ini
        if (board.isValidSolution()) return true;
```

```
        // Mekanisme Increment (Geser Ratu)
        int rowToMove = N - 1;
        while (rowToMove >= 0) {
            int currentCol = board.getState()[rowToMove];

            if (currentCol < N - 1) {
                board.placeQueen(rowToMove, currentCol + 1);
                break; // Berhasil geser, lanjut iterasi berikutnya
            } else {
                board.placeQueen(rowToMove, 0); // Reset & Carry ke atas
                rowToMove--;
            }
        }

        // Jika carry menembus baris 0, ruang pencarian habis
        if (rowToMove < 0) break;
    }
    return false;
}
```

Bab 3. Pengujian dan Bukti Eksekusi

Pengujian dilakukan pada lingkungan sistem operasi **Linux (WSL)** menggunakan antarmuka baris perintah (*Command Line Interface*). Program diuji dengan berbagai jenis *test case*

untuk memastikan kebenaran algoritma, ketahanan (*robustness*) terhadap input invalid, dan performa waktu eksekusi.

Berikut adalah 5 skenario pengujian utama:

3.1. Skenario 1: Solusi Valid (Papan Standar 8x8)

Pengujian dilakukan pada papan berukuran 8×8 dengan wilayah warna yang kompleks.
Input File: `test/tc_8x8_valid.txt`

Analisis: Program berhasil menemukan solusi tunggal yang valid. Visualisasi memperlihatkan posisi Ratu (huruf 'Q' atau blok berwarna) tersebar tanpa ada yang saling serang secara vertikal, horizontal, maupun diagonal, serta mematuhi batasan wilayah warna.

3.2. Skenario 2: Kasus Tanpa Solusi (Impossible Case)

Pengujian dilakukan pada papan ukuran kecil (3×3) yang secara matematis tidak memiliki solusi valid untuk aturan *No-Touching*. **Input File:** `test/tc_3x3_impossible.txt`

```
==========================================
              HASIL EKSEKUSI
==========================================
[FAILED] Tidak ada solusi yang mungkin.
------------------------------------------
Waktu Eksekusi : 0.0825 ms
Total Iterasi  : 27
------------------------------------------
```

Analisis: Algoritma melakukan penelusuran lengkap (*exhaustive search*) ke seluruh ruang kemungkinan dan menyimpulkan secara tepat bahwa tidak ada solusi yang mungkin.

3.3. Skenario 3: Validasi Input (Dimensi Tidak Konsisten)

Pengujian ketahanan parser terhadap file input yang formatnya cacat (bukan persegi). **Input File:** `test/tc_not_square.txt`

```
[INPUT] Masukkan nama file (ketik 'exit' untuk keluar): tc_not_square.txt
[PARSER ERROR] Dimensi tidak konsisten.
```

Analisis: Modul `Parser` berhasil mendeteksi ketidakkonsistenan dimensi baris sebelum algoritma dijalankan, mencegah terjadinya *undefined behavior* atau *crash*.

3.4. Skenario 4: Validasi Karakter Ilegal

Pengujian terhadap file yang mengandung karakter di luar spesifikasi (selain A-Z dan spasi). **Input File:** `test/tc_bad_char.txt`

```
[INPUT] Masukkan nama file (ketik 'exit' untuk keluar): tc_bad_char.txt
[PARSER ERROR] Karakter ilegal '1'.
```

Analisis: Sistem sanitasi input menolak karakter numerik atau simbol yang tidak valid, memastikan data yang masuk ke `Board` selalu bersih.

LAMPIRAN

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

Ega Luthfi Rais
13524115

https://github.com/road2qnt/Tucil1-13524115

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	**V**	
2	Program berhasil dijalankan	**V**	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	**V**	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	**V**	
5	Program memiliki Graphical User Interface (GUI)		**V**
6	Program dapat menyimpan solusi dalam bentuk file gambar		**V**