Laporan Tugas Kecil 1

Penyelesaian Permainan Queens Linkedin

IF2211 Strategi Algoritma

Disusun Oleh

Yusuf Faishal Listyardi (13524014)

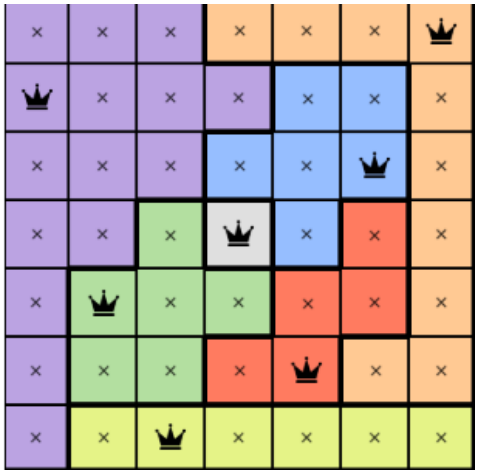
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132

2026

Daftar Isi

1 Deskripsi Masalah

Queens adalah gim logika yang tersedia pada situs jejaring profesional LinkedIn. Game ini lumayan mirip dengan sudoku yang memiliki aturan penempatan khusus pada setiap baris dan kolom.



Gambar 1: Permainan Queens pada Linkedin

Tujuan dari game ini adalah meletakkan *Queens* pada sebuah papan berwarna berukuran N x N atau persegi sehingga memenuhi beberapa syarat berikut :

1. Hanya ada satu *Queens* pada setiap baris dan kolom
2. Hanya ada satu *Queens* pada setiap warna
3. Tidak boleh ada 2 *Queens* yang diletakkan bersebelahan baik secara baris, kolom ataupun diagonal

Tujuan dari program ini mencari **cukup satu** solusi untuk menempatkan *Queens* pada papan dengan menggunakan algoritma **Brute Force**, atau menampilkan jika tidak ada solusi yang memenuhi dari sebuah konfigurasi papan.

2 Algoritma Penyelesaian

2.1 Algoritma Brute Force

Algoritma brute force merupakan metode pencarian dasar yang menyeluruh, di mana semua kemungkinan solusi diperiksa satu per satu sampai menemukan penyelesaian dari suatu permasalahan. Pendekatan ini menyelesaikan masalah dengan cara yang sederhana, lugas, jelas, dan mudah untuk dimengerti. Metode ini juga kerap dikenal dengan sebutan algoritma naif (naive algorithm).

Algoritma brute force umumnya tidak "cerdas" dan tidak sangkil, karena ia membutuhkan cost komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Oleh karena itu, algoritma brute force lebih cocok untuk persoalan yang ukuran masukannya (n) kecil.

Algoritma ini memiliki banyak kelebihan salah satunya adalah implementasi yang cenderung sederhana dan mudah dibanding algoritma lain. Selain itu kelebihan utama algoritma ini adalah dapat diterapkan untuk memecahkan semua persoalan (wide applicability) (Rinaldi, 2026).

2.2 Exhaustive Search

Exhaustive search adalah algoritma yang tergolong algoritma brute force, dimana kita mencoba semua kemungkinan solusi yang ada satu per satu sampai ditemukan solusi yang benar. Langkahnya sangat sederhana yaitu :

1. Tentukan semua kemungkinan solusi secara sistematis
2. Evaluasi setiap kemungkinan solusi satu per satu, apakah memenuhi syarat atau tidak.
3. Bila pencarian berakhir, umumkan solusi jika ada

Meskipun exhaustive search secara teoritis menghasilkan solusi, namun waktu atau sumberdaya yang dibutuhkan dalam pencarian solusinya sangat besar (Rinaldi, 2026).

Karena pada spesifikasi tugas kecil dikatakan untuk menggunakan algoritma brute force murni. Sehingga, pada tugas kecil ini saya memutuskan untuk menggunakan algoritma **brute force dengan exhaustive search**. Untuk penjelasan lebih lanjut ada di bagian 2.3.

2.3 Abstraksi dan Langkah Penyelesaian

2.3.1 Kemungkinan Solusi

Sesuai dengan langkah pertama pada exhaustive search, kita tentukan semua kemungkinan solusi secara sistematis, karena ada beberapa syarat yang harus kita penuhi, kita bisa reduksi kemungkinan solusi adalah kemungkinan konfigurasi penempatan *Queens* yang memenuhi syarat bahwa setiap baris dan kolom hanya memiliki satu *Queens* atau konfigurasi yang memenuhi **syarat pertama**.

Misal papan berukuran $N \times N$. Kita bisa menaruh *Queens* pada baris 1 dengan N kemungkinan. Lalu untuk baris 2 ada $N - 1$ kemungkinan peletakan, karena satu kemungkinan sudah dipakai baris 1. Ini berlanjut hingga pada baris N yang hanya memiliki 1 kemungkinan peletakan *Queens*. Sehingga total kemungkinan solusi yang kita coba adalah $N \times N - 1 \times \cdots \times 1 = N!$ kemungkinan, dan program memiliki kompleksitas $O(N!)$.

2.3.2 Evaluasi Kemungkinan

Untuk setiap kemungkinan solusi hanya syarat 2 dan 3 yang belum terpenuhi, karena itu untuk setiap kemungkinan kita cek apakah ada 2 *Queens* yang memiliki warna sama dan juga apakah ada 2 *Queens* yang bersebelahan secara diagonal, baris, dan kolom.

Jika pada saat pengecekan ada salah satu syarat yang tidak terpenuhi, maka pengecekan akan dilanjutkan untuk permutasi atau kemungkinan selanjutnya. Proses ini berulang sampai ada kemungkinan yang lolos pengecekan semua syarat atau sampai semua kemungkinan sudah diiterasi.

3 Implementasi

3.1 Kemungkinan Solusi

```cpp
vector<int> p(N);
for (int i = 0; i < N; i++)
{
    p[i] = i;
}

bool found = false;

auto mulai = high_resolution_clock::now();
do
{
    if (valid(p, grid, N))
    {
        found = true;
        break;
    }
} while (next_permutation(p.begin(), p.end()));
```

Gambar 2: Kode Program Kemungkinan Solusi

Pertama, dibuat sebuah array P dengan ukuran N (ukuran dari papan). Disini indeks array mewakili nomer baris dan nilai array mewakili nomer kolom. Misalkan kita punya $P[1] = 3$, maka pada baris 1 *Queens* diletakkan di kolom ke 3. Pada awalnya, kita isi array P secara berurutan yaitu $1, 2, \ldots, N$.

Selanjutnya kita akan mencoba semua kemungkinan permutasi yang ada, atau kita ubah urutan dari isi array awal tadi. Pada langkah ini kita menggunakan fungsi bawaan c++ yaitu next_permutation. Fungsi ini bekerja dengan cara mengubah urutan elemen di dalam array menjadi urutan berikutnya berdasarkan urutan leksikografis (seperti urutan kata di kamus). Untuk setiap kemungkinan permutasi, kita cek apakah memenuhi syarat 2 dan 3. Ini akan dijelaskan di bagian 3.2.

Karena kita hanya merubah urutan bukan merubah isi array, maka tidak akan pernah ada angka yang sama di dalam array tersebut. Kalau isinya tidak pernah sama, berarti tidak akan pernah ada 2 *Queens* yang berada di kolom yang sama.

3.2 Pengecekan Solusi Valid

1. Cek Warna

   Untuk mengecek apakah satu warna hanya diisi satu *Queens*, penulis menggunakan array Color yang jika isinya $True$ maka sudah ada *Queens* yang mengisi color tersebut, sebaliknya jika isinya adalah $False$.

   Jika pada saat iterasi ditemukan bahwa color *Queens* saat ini sudah terisi, maka dapat disimpulkan konfigurasi ini tidak valid.

2. Cek Bersebelahan

   Karena konfigurasi kemungkinan solusi menjamin tidak ada 2 *Queens* yang berada pada baris dan kolom yang sama, itu juga menjamin tidak ada 2 *Queens* yang bersebelahan secara baris dan kolom, sehingga kita hanya perlu mengecek apakah ada 2 *Queens* yang bersebelahan secara diagonal.

   Pengcekan bisa dilakukan dengan menghitung jarak baris dan kolom dari 2 *Queens*, jika kedua jaraknya adalah 1, maka 2 *Queens* dinyatakan bersebalahan secara diagonal dan konfigurasi dinyatakan tidak valid.

Jika sebuah konfigurasi lolos dari 2 pengecekan diatas, maka sebuah konfigurasi dinyatakan sebagai salah satu solusi valid yang memenuhi semua syarat, dan proses dihentikan langsung.

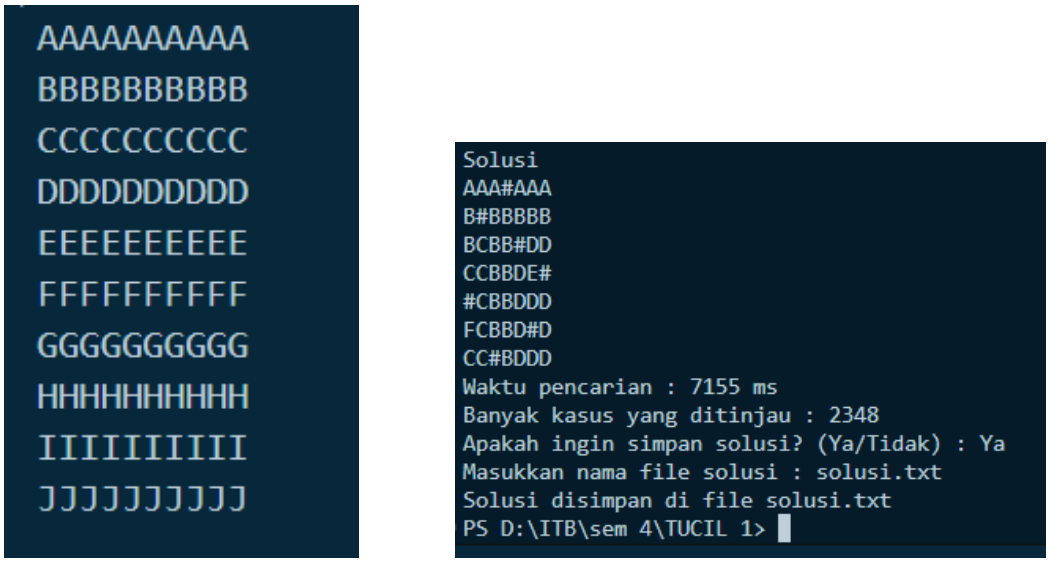
```cpp
bool valid(vector<int> &p, vector<vector<char>> &grid, int N)
{
    vector<bool> color(26, false);
    cnt++;

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

        // cek warna
        int warna = grid[r][c] - 'A';
        if (color[warna])
            return false; // tidak memenuhi syarat satu warna satu queen
        color[warna] = true;

        // cek bersentuhan diagonal apa engga
        for (int j = 0; j < i; j++)
        {
            int k = p[j];

            if (abs(r - j) == 1 and abs(c - k) == 1)
            {
                return false;
            }
        }
    }
    return true;
}
```

Gambar 3: Kode Program Pengecekan Solusi Valid

4 Eksperimen

4.1 Test Case 1

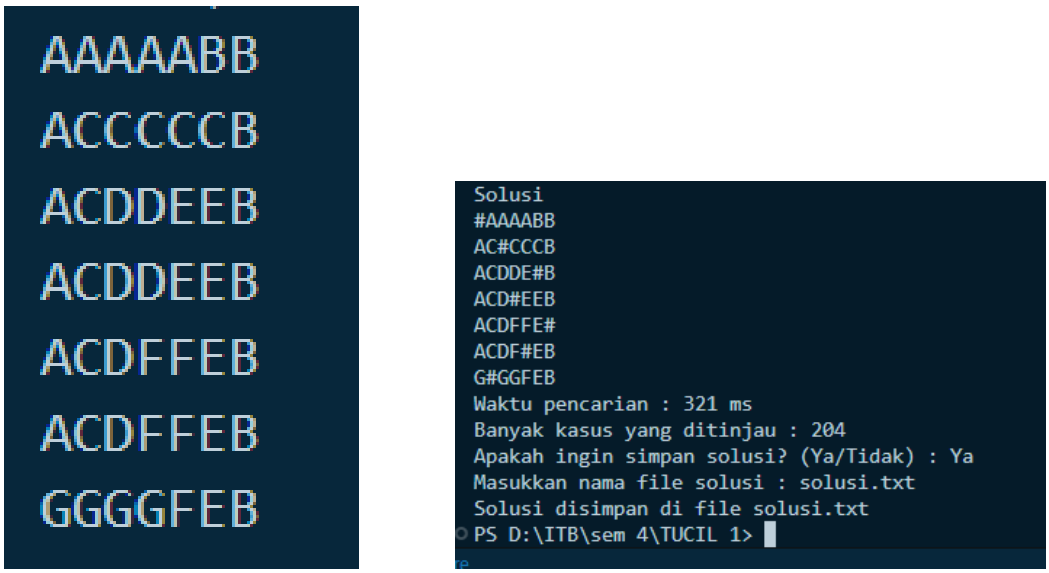
Test Case berupa papan 10×10 yang memiliki solusi



Gambar 4: Input (kiri) dan Output (kanan) Test Case 1

4.2 Test Case 2

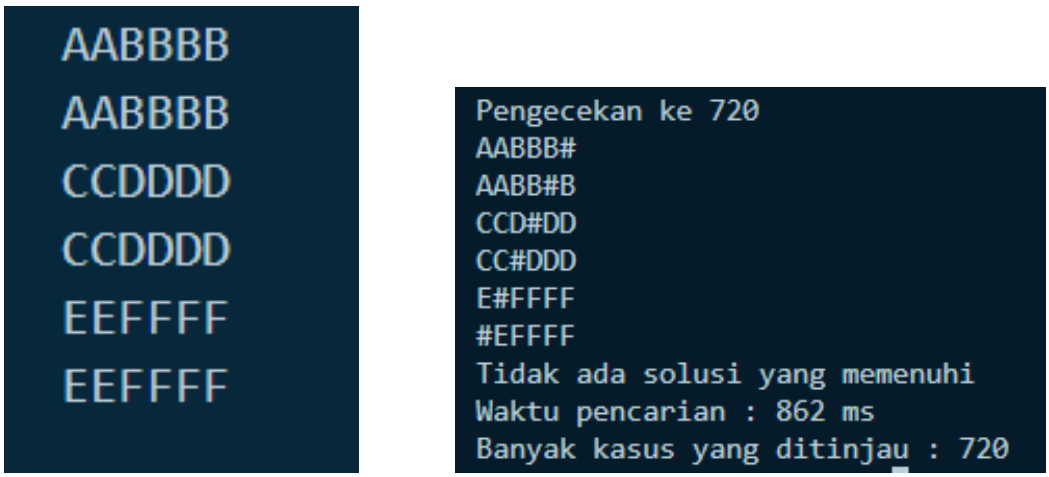
Test Case berupa papan 7×7 yang memiliki solusi



Gambar 5: Input (kiri) dan Output (kanan) Test Case 2

4.3 Test Case 3

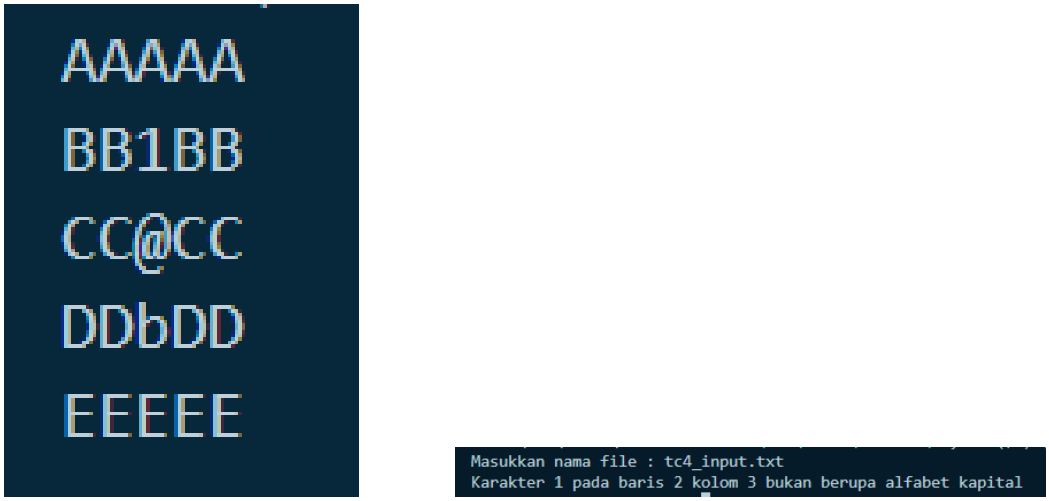
Test Case berupa papan 6×6 yang tidak memiliki solusi



Gambar 6: Input (kiri) dan Output (kanan) Test Case 3

4.4 Test Case 4

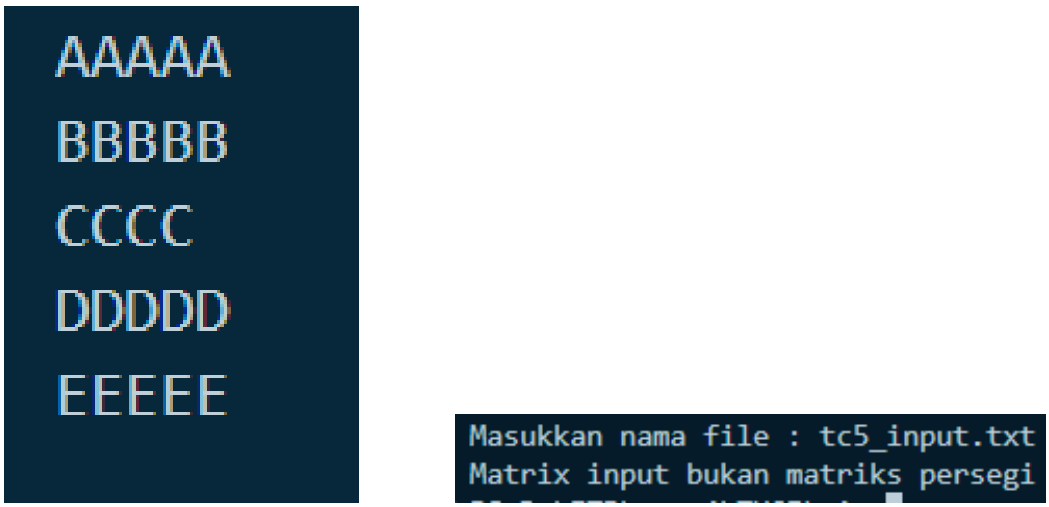
Test Case invalid berupa isi papan bukan huruf alphabet



Gambar 7: Input (kiri) dan Output (kanan) Test Case 4

4.5 Test Case 5

Test Case invalid berupa papan yang bukan berukuran $N \times N$



Gambar 8: Input (kiri) dan Output (kanan) Test Case 5

5 Penutup

5.1 Tautan

Repository program dapat diakses melalui tautan berikut:

`https://github.com/ucupsss/Tucil1_13524014.git`

5.2 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		✓

5.3 Pernyataan Penggunaan Generative AI

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

Yusuf Faishal Listyardi

Referensi

Rinaldi Munir. 2026. "Algoritma Brute Force (Bagian 1) (Versi baru 2026)." `https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/02-Algoritma-Brute-Force-(2026)-Bag1.pdf`

Geeks for Geeks. 2026. `std::next_permutation` and `std::prev_permutation` in C++. Diakses pada 16 Februari 2026. `https://www.geeksforgeeks.org/cpp/stdnext_permutation-prev_permutation-c/`