LAPORAN TUGAS KECIL 1

IF 2211 SRATEGI ALGORITMA

PENYELESAIAN PERMAINAN QUENSS LINKEDIN

DISUSUN OLEH :

MANUEL THIMOTY SILALAHI / 13524102

Bab I. Pendahuluan

Permainan Queens LinkedIn merupakan salah satu puzzle yang terkenal di LinkedIn. Dalam permainan ini, pemain diminta untuk menempatkan Quenn (Ratu) pada sebuah papan berukuran $n \times n$. Setiap kotak memiliki warna tertentu dan berapa kotak memiliki warna yang sama.

Aturan dalam penempatan quenn adalah sebagai berikut :

1. Hanya boleh terdapat tepat 1 Queen pada setiap baris.
2. Hanya boleh terdapat tepat 1 Queen pada setiap kolom.
3. Hanya boleh terdapat tepat 1 Queen pada setiap warna.

Bab II. Observasi

Salah satu konsekuensi dari aturan permainan ini adalah jumlah kolom, jumlah baris, dan jumlah warna unik haruslah sama. Misalkan banyak baris = M, banyak kolom = N, dan banyak warna unik = D. Secara matematis, sebuah konfigurasi mempunyai solusi jika dan hanya jika

$$M = N = D$$

Dan memenuhi ketiga aturan pada Bab I.

Salah satu metode yang dapat digunakan adalah *Brute Force*. Dalam praktiknya, metode ini tentu sangat tidak efisien, terutama untuk kasus nilai N yang sangat besar. Namun, ide yang dibutuhkan untuk mencapai metode ini tentunya paling mudah daripada metode lainnya.

Metode bruteforce yang digunakan adalah sebagai berikut :

1. Untuk setiap baris, letakkan tepat 1 Quenn di kolom manapun. Peletakan dimulai dari baris paling atas.
2. Ketika telah mencapai baris paling bawah (atau jumlah Queen yang telah diletakkan sebanyak N), lakukan pengecekan apakah penempatan tersebut memenuhi ketiga aturan pada Bab I.
3. Jika memenuhi, maka solusi telah ditemukan. Jika tidak, pindahkan Queen terakhir ke kolom lain yang memungkinkan.
4. Jika pada baris tersebut sudah tidak ada opsi kolom yang tersedia, lakukan backtracking ke baris sebelumnya dan ubah posisi Queen di baris tersebut.
5. Ulangi langkah 1–4 hingga ditemukan solusi yang valid atau hingga dipastikan tidak ada solusi.

Secara sekilas, algoritma atau cara berpikir ini terlihat naif, terlalu *obviuous* dan sederhana. Namun, waktu yang dibutuhkan untuk memperoleh solusi pada ukuran papan tertentu masih tergolong wajar, serta metode ini relatif mudah untuk diimplementasikan.

Bab III. Implementasi

Berikut adalah pseudocode implementasi dari prosedur penyelesaian masalah utama.

Algorithm 2 Algoritma Brute-Force Penyelesaian Masalah Queen

```
1: procedure SOLVE(x)
2:     if x = N then
3:         queens ← Extract all placed queens
4:         if ValidateSolution(queens) then
5:             found ← True
6:             Output solutions grid
7:         end if Return
8:     end if
9:     for col = 0 to n − 1 do
10:        if not found then
11:            used[row][col] ← True
12:            Solve(x + 1)
13:            used[row][col] ← False
14:        end if
15:    end for
16: end procedure
```

Berdasarkan pseducode diatas, pertama – tama kita akan memanggil $Solve(0)$, yaitu kita muali mengisi dari baris ke – 0. Kemudian, kita akan loop setiap kolom ke – 0 sampai ke n-1. Dan selama kita belum menemukan solusi , kita akan tandai input di baris ke-*row* dan kolom ke-*col* sebagai true, menandakan telah digunakan. Kemudian, kita akan jalankan $Solve(x + 1)$ yaitu mengisi baris ke – 2.

Hal ini akan kita ulangi terus, sampai di kasus nilai $x = N$, Dimana N adalah ukuran grid. Di kondisi ini, semua baris sudah terisi oleh tepat 1 Queen. Kemudian kita akan memvalidasi solusi sekarang. Validasi dilakukan dengan mengecek setiap queen satu persatu apakah sesuai dengan 3 aturan utama di bab sebelumnya. Jika iya, maka kita berhasil menemukan solusi. Namun, jika tidak, misalkan sekarang di state(baris) ke k, maka kita akan mereturn $Solve(k)$. Kemudian, kita akan mundur 1 langkah ke kondisi dimana $Solve(k)$ dipanggil, yakni saat state $Solve(k - 1)$. Maka, queen di row dan col saat prosedur tersebut dipanggil diubah ke False, atau dalam hal ini tidak ditaruh disitu.

Algoritma ini terus berulang – ulang sampai ditemukan 1 solusi, atau dapat dipastikan bahwa tidak ada satupun solusi yang memenuhi. Kompleksitas dari menggenerate posisi sampai ke kondisi akan di return (sampai $x = N$) adalah $O(n^n)$ maksimum. Sementara kompleksitas pada fungsio validasi adalah $O(n^2)$ maksimum. Maka, kompleksitas worst case akhir dari fungsi $Solve(x)$ adalah $O(n^2 \cdot n^n)$.

Bab IV. Source Code dan Hasil

Berikut adalah Kode yang digunakan dalam implementasi algoritma. Bahasa yang dipilih adalah C++. Kode dibagi menjadi 2, native.cpp yang diimplementasikan menggunakan exhaustive search. Sementara itu, ada optimize.cpp yang diimplementasikan dengan pruning lebih cepat. Naive.cpp memiliki

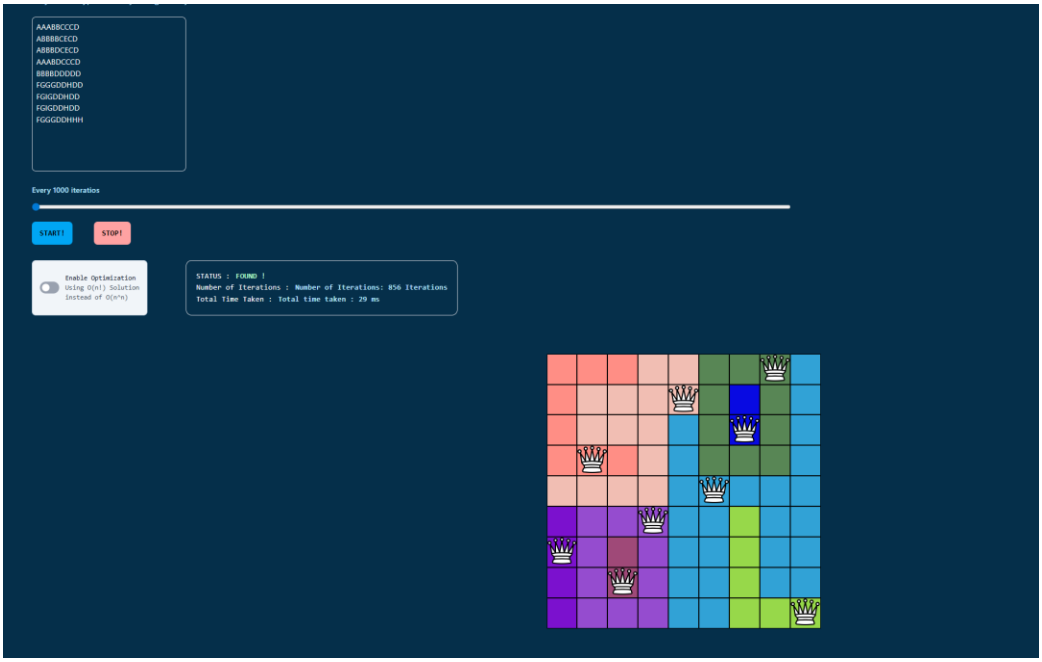
kompleksitas teoritis $O(n^n)$ sementara optimize.cpp $O(n!)$. yang diletakkan di laporan ini hanyalah kode Naive. cpp yang mengimplementasikan bruteforce secara murni.

```
        }
        if(found){
            solusi = true;
            for(int i = 0 ; i < n ; i++){
                for(int j = 0 ; j < n ; j++){
                    if(used[i][j]){
                        cout << "#";
                    }
                    else{
                        cout << input[i][j];
                    }
                }
                cout << endl;
            }
            return;
        }
        return;
    }

    for (int i = 0; i < n; i++){
        if (solusi) return;


        used[x][i] = true;
        solve(x + 1);
        used[x][i] = false;
    }
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> n;

    for (int i = 0; i < n; i++){
        string s;
        cin >> s;
        for (int j = 0; j < n; j++){
            input[i][j] = s[j];
            used[i][j] = false;
        }
    }
    cin >> every;
    auto start =  std::chrono::high_resolution_clock::now();
    solve();
    auto end = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - start);
    cout << endl;

    cout << it << endl;
    cout.flush();

    auto ms_duration = std::chrono::duration_cast<std::chrono::milliseconds>(end - start);
    cout << ms_duration.count() - countsleep * sleepduration << endl;
    cout.flush();

    if (!solusi)
        cout << 0 << endl;
    else{
        cout << 1 << endl;
    }
    cout.flush();
    return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;

char input[100][100];
bool used[100][100];

int n;
bool solusi = false;
int it = 0;
int countsleep = 0;
int sleepduration = 100;
int every = 0;
bool cek(int x, int y){
    char warna = input[x][y];

    for (int i = 0; i < x; i++){
        for (int j = 0; j < n; j++){
            if (used[i][j]){
                if (warna == input[i][j]) return false;

                if (j == y) return false;
                if(x == i) return false;
                if (abs(x - i) == 1 && abs(y - j) == 1)
                    return false;
            }
        }
    }
    return true;
}

void solve(int x = 0){
    it ++;
    if(solusi) return;
    if(it % every == 0){
        for(int i = 0 ; i < n ; i++){
            for(int j = 0 ; j < n ; j++){
                if(used[i][j]) cout << "#";
                else cout << input[i][j];
            }
            cout << endl;
        }
        cout << endl;
        cout.flush();
        this_thread::sleep_for(chrono::milliseconds(sleepduration));
        countsleep++;
    }
    if (x == n){
        // solusi = true;
        bool found = true;
        vector<pair<int,int>> queens;
        for (int i = 0; i < n; i++){
            for (int j = 0; j < n; j++){
                if(used[i][j]){
                    queens.push_back(make_pair(i,j));
                }
            }
        }
        for(int i = 0 ; i < n ; i++){
            for(int j = i+1 ; j < n ; j++){
                int x1 = queens[i].first;
                int x2 = queens[j].first;
                int y1 = queens[i].second;
                int y2 = queens[j].second;
                if(x1 == x2 || y1 == y2) found = false;
                if(input[x1][y1] == input[x2][y2]) found = false;
                if(abs(x1-x2) == 1 && abs(y1-y2) == 1) found = false;
                if(!found) break;
            }
```

Berikut adalah hasil dari eksekusi program dan GUI :

1. Input :
   AAABBCCCD
   ABBBBCECD
   ABBBDCECD
   AAABDCCCD
   BBBBDDDDD
   FGGGDDHDD
   FGIGDDHDD
   FGIGDDHDD
   FGGGDDHHH



2. Input :
   ABC
   BAC
   BCA

3. Input :

AAAAAAA

AABADDA

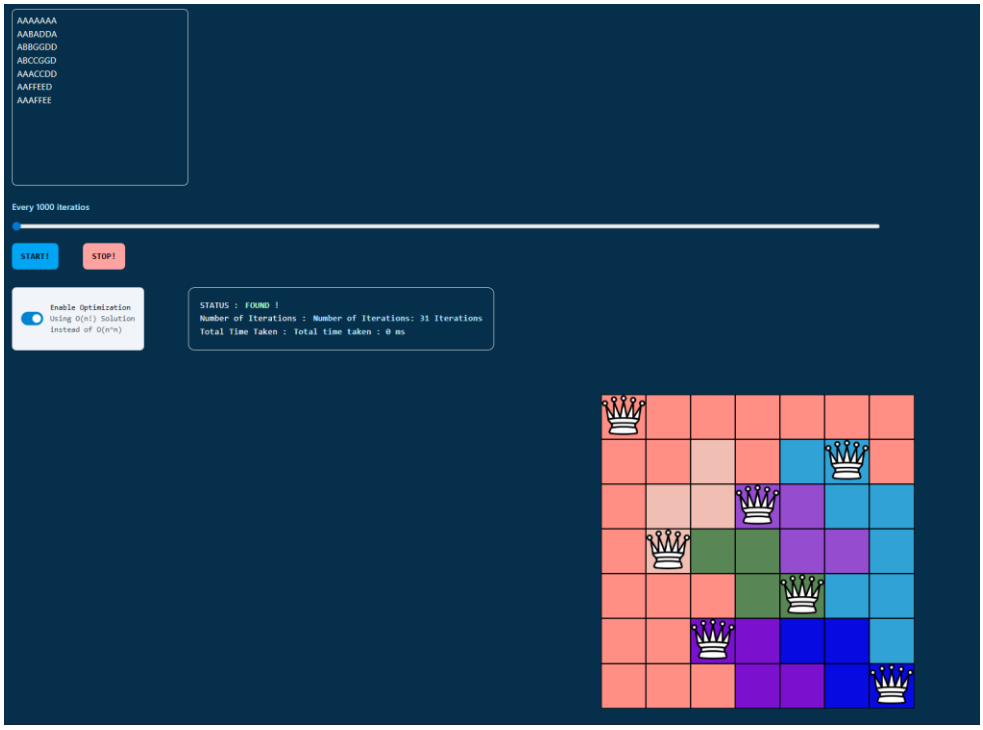
ABBGGDD

ABCCGGD

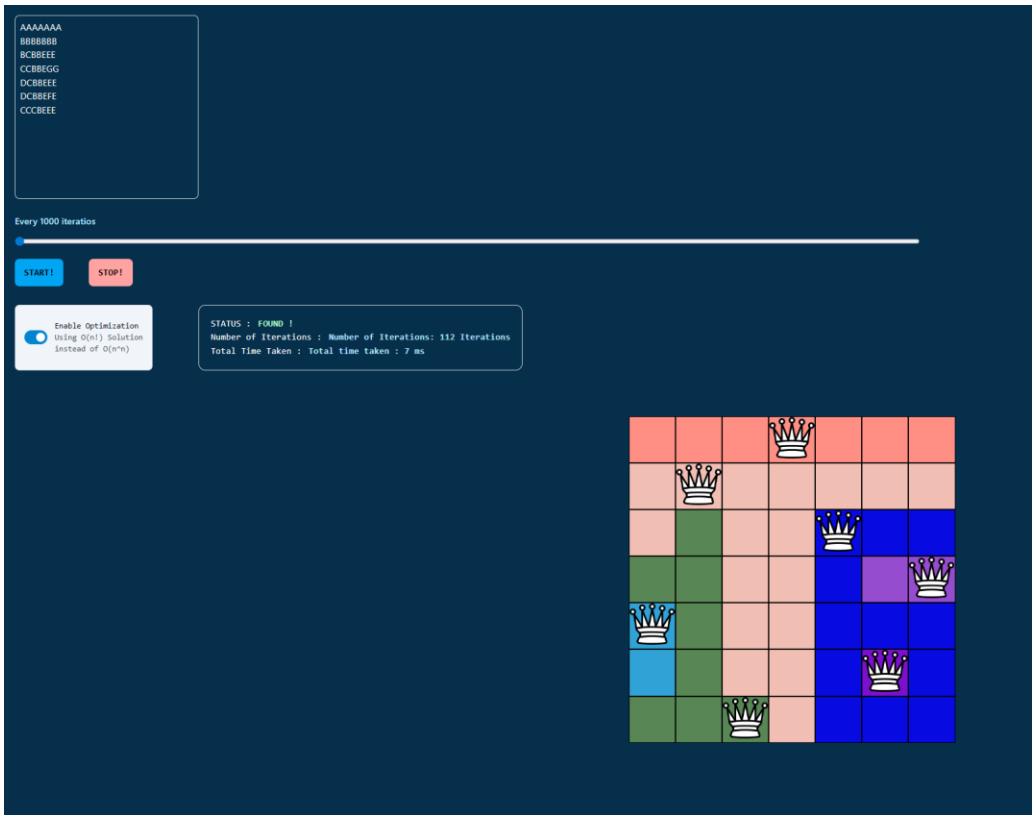
AAACCDD

AAFFEED

AAAFFEE



4. Input :
   AAAAAAA
   BBBBBBB
   BCBBEEE
   CCBBEGG
   DCBBEEE
   DCBBEFE
   CCCBEEE


AAAAAAA
BBBBBBB
BCBBEEE
CCBBEGG
DCBBEEE
DCBBEFE
CCCBEEE
Every 1000 iteratios
START!
STOP!
Enable Optimization
Using O(n!) Solution
instead of O(n^n)
STATUS : FOUND !
Number of Iterations : Number of Iterations: 112 Iterations
Total Time Taken : Total time taken : 7 ms

5. Input : A


A
Every 506369 iterations
START!
STOP!
Enable Optimization
Using O(n!) Solution
instead of O(n^n)
STATUS : FOUND !
Number of Iterations : Number of Iterations: 2 Iterations
Total Time Taken : Total time taken : 22 ms

Link Tautan GitHub : https://github.com/manuellthimoty/tucil-stima-1

Pernyataan Penggunaan AI

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

Manuel Thimoty Silalahi

13524102