LAPORAN TUCIL 1 STIMA

LAPORAN TUGAS KECIL 1 “QUEENS”

Strategi Algoritma - IF2211

Billie Bhaskara Wibawa

13524024

17.02.2026

GARIS BESAR PROYEK

Pengerjaan tugas kecil pertama mata kuliah Strategi Algoritma “QUEENS” berikut saya kerjakan menggunakan C dan Python. Penyambungan Python dengan C dilakukan dengan *library ctypes* yang mana memungkinkan untuk *passing returns* dari fungsi C. Selain itu, *library* ini juga dapat memungkinkan *passing* fungsi Python ke C sebagai *pointer* yang dapat dipanggil oleh C.

Dengan semua yang telah disebut, pada akhirnya lapisan Python hanya untuk *input/output* processing (*image* dan *text)* serta GUI.

PENJELASAN ALGORITMA

PENJELASAN ALGORITMA TANPA *EARLY PRUNING*

Agar laporan ini mudah dibaca saya akan mengambil potongan-potongan kode dari *source code* yang terletak pada *folder src* pada *repository*. Direkomendasikan untuk juga membuka *source code*-nya selagi membaca bagian ini (src/queens_logic.c).

```
static int solve_slow_recursion(int row, int size, int
solution[size][size], char board[size][size]){
    // Base Case
    if (row == size) {
        if (is_whole_board_valid(size, solution, board)) {
            return 1;
        }
        return 0;
    }
    int result;
    for (int col = 0; col < size; col++){
        solution[row][col] = 1;
        result = solve_slow_recursion(row + 1, size, solution, board);
```

```
        if (result == 1) {
            return 1;
        }
        else if (result == -1){
            return -1;
        }

        // Backtrack
        solution[row][col] = 0;
    }
    return 0;
}
```

Algoritma yang digunakan adalah algoritma rekursif tanpa *early pruning*. Artinya semua pengecekan validitas solusi dilakukan setelah diletakkan *N* ratu.

Dalam algoritma rekursi ini, tiap ratu sudah mulai di masing-masing baris sebagai bentuk optimisasi kecil.

Algoritma rekursi yang digunakan sederhana dengan langkah-langkah berikut:

1. Cek apakah sudah mencapai *base case* di mana *row==size* yang artinya sudah dilakukan traversal sampai akhir. Apabila iya, maka cek apakah solusi valid. Jika iya kembalikan 1, jika tidak kembalikan 0.
2. Dalam bagian rekursi, untuk tiap baris dijalankan *for-loop* sendiri untuk mencoba meletakkan ratu di masing-masing kolom yang ada.
3. Apabila didapat 0 dari pengembalian hasil rekursi, artinya solusi tidak berhasil. Dilakukan backtrack dengan menghapus ratu di kolom yang telah dipilih, dan dilanjut dengan meletakkan queen pada kotak selanjutnya pada iterasi *for-loop* selanjutnya.
4. Apabila didapat 1 dari pengembalian hasil rekursi, artinya solusi berhasil. Dengan menerima 1, fungsi rekursi langsung melanjutkan *return* 1, sehingga seluruh rantai rekursi berakhir dan diberikan solusi terakhir.

PENJELASAN ALGORITMA DENGAN *EARLY PRUNING*

```
int solve_queens(args...){
    // Setups…
    int occupied_region[256];
    int result = solve_queen_recursion(args...);
    return result;
}
```

Kutipan kode di atas adalah *entry-point* dari algoritma *brute force* yang saya gunakan. Algoritma ini memerlukan entry point karena algoritma ini menggunakan metode rekursif. Saya menggunakan metode rekursif karena menggunakan pendekatan *state-machine* atau algoritma *loop-based* memerlukan banyak variabel untuk *tracking* posisi *queen* yang telah diletakkan.

Pendekatan rekursif sangat memudahkan karena sebagian besar *state* yang harus diingat sudah diatasi oleh *call stack.*

Terakhir dalam kutipan kode di atas ada *array occupied_region.* Fungsi dari *array* ini adalah untuk memeriksa apakah suatu daerah sudah ada *queen*-nya. Cara kerja *array* ini mirip set di mana apabila *queen* diletakkan pada zona ‘A’ sebagai contoh, maka occupied_region[‘A’] akan di-*set* menjadi 1, menandakan sebuah *queen* telah memenuhi daerah itu. Ukuran dari *array* ini 256 karena *range* dari *extended ASCII* yang memiliki karakter dengan indeks 0-255. Hal ini juga dikarenakan dalam implementasi saya, zona warna direpresentasikan sebagai karakter ASCII.

```
int solve_queen_recursion(args...){
    /* == LIVE UPDATE LOGIC == */
    g_iteration_count++;
    if (interrupt_func != NULL && g_update_freq > 0) {
        if (g_iteration_count % g_update_freq == 0) {
            // Cast 2D array to flat pointer for Python
            if(!interrupt_func(size, (int*)solution)){
```

```
                return -1; // To stop the recursion dead in its tracks and
return a special code (-1) for error in python

            }

        }

    }

    int result;

    // Base cases

    if(row==size) return 1;

    else if (col==size) return 0;



    if(is_safe(args...)){

        occupied_region[(unsigned char)board[row][col]] = 1;

        solution[row][col] = 1;

        result = solve_queen_recursion(row+1, 0, size, solution, board,
occupied_region);

    }

    else return solve_queen_recursion(row, col+1, size, solution, board,
occupied_region);



    if(result==1){

        return 1;

    }

    else if(result == -1){

        return -1;

    }

    else{

        // Traceback and cleaning up the footprints (solution matrix and
occupied regions)

        solution[row][col] = 0;

        occupied_region[(unsigned char)board[row][col]] = 0;
```

```
        return solve_queen_recursion(row, col+1, size, solution, board,
occupied_region);
    }
}
```

Sebagian besar algoritma terjadi di sini. Karena spesifikasi tugas hanya membolehkan untuk algoritma yang secara murni *bruteforce,* maka rekursi juga cenderung sederhana. Sebagian besar logika ada pada bagian:

```
    if(result==1){
        return 1;
    }
    else if(result == -1){
        return -1;
    }
    else{
        solution[row][col] = 0;
        occupied_region[(unsigned char)board[row][col]] = 0;
        return solve_queen_recursion(row, col+1, rest of args...);
    }
}
```

Dalam bagian ini ditunjukkan bagaimana algoritma rekursif ini bercabang. Apabila *result* == 1, maka rekursi berhasil dan langsung return 1 untuk menghabiskan rantai rekursi. Namun apabila result = 0 (ditandai pada bagian *else)* maka artinya jalur rekursi yang telah dilakukan (untuk *row* dan *col* yang diberikan) tidak berhasil dan telah *backtrack* ke titik sekarang. Yang mana akhirnya dilakukan lagi rekursi tetapi iterasi selanjutnya dilakukan ke kolom yang depan (ditandai dengan *row* yang tetap, tetapi *col* menjadi *col+1).*

Terakhir, pengecekan *result* == -1 dilakukan apabila terjadi kesalahan dalam memanggil fungsi *callback* ke Python agar untuk segera menyelesaikan rekursi dan menghentikan

semua proses. Intinya pengecekan ini hanya untuk *layer* Python memberikan interupsi terjadi *error* dan menghentikan *layer* C juga.

Kesimpulannya, berikut cara kerja algoritma ini:

1. Rekursi dimulai pada baris 0 dan kolom 0
2. Rekursi kemudian berjalan mencari kolom pertama yang aman untuk diletakkan sebuah ratu. Perjalanan ini selalu dari kolom 0 dan berjalan naik (dalam gambar visual, maka berjalan ke kanan)
3. Apabila sudah ditemukan kolom pertama yang aman, maka algoritma lanjut ke baris setelahnya dan melakukan perjalanan kolom dari kolom 0 lagi
4. Kemudian terjadi percabangan tergantung *ending* yang dicapai. Apabila mencapai row==size, maka artinya sudah mencapai barus akhir (size-1) dan kemudian dijalankan sebuah rekursi sekali lagi (size-1+1 menjadi size). Maka artinya telah dicapai *ending* yang sukses dan mengembalikan 1, yang mana akan menyelesaikan semua rekursi di bawahnya
5. Apabila ending yang dicapai adalah col==size, artinya perjalanan kolom telah mencapai kolom terakhir (size-1) dan masih lanjut ke kolom selanjutnya (size-1+1 menjadi size). Artinya dicapai *ending* di mana perjalanan kolom sudah mencapai ujung, tetapi masih belum ada kotak yang memuaskan peraturan *queens* dan lanjut mencari ke kolom di luar *boundary*. Ini merupakan *ending* yang gagal dan fungsi rekursi menghasilkan nilai 0, yang mana akan diterima fungsi rekursi pada iterasi sebelumnya, dan fungsi rekursi itu akan lanjut sesuai branching yang ada. Dalam kasus ini, saat menerima nilai 0, branching yang dilakukan adalah untuk menjalankan fungsi rekursi terhadap baris yang sama tetapi dilakukan pada kolom selanjutnya (col+1)
6. Apabila tidak ditemukan solusi, maka fungsi rekursi pertama akan mengembalikan nilai 0 yang kemudian diterima oleh fungsi *entry*.

DETAIL TEKNIS LAINNYA

Bagian ini berfungsi untuk menjelaskan implementasi teknis non-algoritma. Dalam konteks proyek ini, maka bagian ini menjelaskan bagaimana saya melakukan optimisasi C. Semua optimisasi hanya ke arah non-algoritma dalam penyelesaian *queens*, seperti data structures yang optimal, atau pengecekan validitas *board* sebelum dilakukan *solve*. Bagian ini juga menjelaskan bagaimana saya menerjemahkan lapisan C ke lapisan Python.

VALIDASI JUMLAH BARIS DAN KOLOM

Akan saya jelaskan beberapa validasi input secara singkat. Pengecekan validasi input file .txt dilakukan dalam layer Python.

```
# Dimension calculations
row_length = content.find('\n')
string_length = len("".join(content.split()))
if(string_length!=(row_length**2)):
    raise ValueError("Each row must have the same number of characters!")
for i in range(string_length):
    if((i%(row_length+1)==row_length) and content[i]!='\n'):
        raise ValueError("Each row must have the same number of
characters!")
self.grid_size = row_length
```

Dalam algoritma di atas, terlihat bahwa panjang baris diambil dengan patokan posisi *newline* pada baris pertama *(content.find('\n')).* Setelah itu, diambil juga panjang seluruh *file* tanpa ada *whitespace.*

Validasi pertama dicek apakah panjang seluruh karakter merupakan panjang baris pertama dikuadratkan. Karena pasti apabila setiap baris memiliki jumlah karakter yang sama dan jumlah baris sama dengan panjang baris pertama, jumlah seluruh karakter dan panjang baris pertama akan sama.

Validasi kedua dilakukan dalam *for-loop* di mana setiap barus dicek apakah panjangnya sama dengan baris pertama.

VALIDASI WARNA

Dalam validasi warna, ada 2 validasi yang dilakukan. Pertama adalah validasi apakah ada pulau warna yang tidak tersambung dengan pulau lainnya. Validasi ini diterapkan menggunakan *BFS.* Validasi kedua adalah validasi apakah jumlah warna sama dengan jumlah baris dan kolom.

```
/**
```

```
 * @brief Check if there is any disconnected color islands using BFS :/
 * @return 1 If there is no disconnected islands, 0 if yes. Oh, also -1 if jumlah color != row or col
 */
static int check_board_islands(int size, char board[size][size]){
    int is_region_checked[256]; // Same as region occuped on solve_queen() but specialized for this
    int color_num = 0;
    unsigned char current_region = '\0';

    for(int i = 0; i<256; i++){ // Memset as usual
        is_region_checked[i] = 0;
    }
    // Looks horrendous with 4 for-loops but will actually be pretty optimized since it'll skip
    // color islands that are already checked
    for(int i = 0; i<size; i++){
        for(int j = 0; j<size; j++){
            if(!is_region_checked[(unsigned char)board[i][j]]){
                color_num++;
                current_region = (unsigned char)board[i][j];
                for(int k = i; k<size; k++){
                    int l = (k==i) ? (j+1):0;
                    for(l; l<size; l++){
                        if(board[k][l]==current_region){
```

```
                            Node start = {i,j};
                            Node goal = {k,l};
                            int result =BFS(start, goal, size, board);
                            if(!result) return 0; // Found a disconnected
color island
                        }
                    }
                }
                is_region_checked[current_region] = 1;
            }
        }
    }
    if(color_num!=size){
        return -1;
    }
    return 1;
}
```

Kutipan kode di atas merupakan algoritma pengecekan pulau warna. Algoritma yang dilakukan adalah untuk menjalankan traversal *board* hingga menemukan warna yang belum diperiksa. Apabila menemukan warna ini, maka dilakukan traversal ke sisa dari board untuk memeriksa semua warna yang sama, dan dilakukan *BFS* untuk ke warna yang sama itu. Dalam *BFS*, traversal hanya boleh dilakukan melalui warna yang sama. Warna lain bertindak sebagai tembok. Apabila ada warna yang tidak bisa dicapai oleh warna yang sama, maka artinya terdapat pulau warna yang terpisah.

Setelah dilakukan *BFS* untuk semua *node* dari suatu warna, maka tidak akan dilakukan lagi *BFS* pada warna itu. Sehingga maksimal pengecekan hanya dilakukan untuk papan dengan 26 warna.

```c
typedef struct {
    int row;
    int col;
} Node; // Node for BFS representing {row, col}
typedef struct Queue {
     Node arr[MAX_POSSIBLE_SQUARE];
     int head;
     int tail;
} Queue;
```

Penerapan Queue sangat sederhana. Queue hanya satu arah (tidak melingkar) dengan MAX_POSSIBLE_SQUARE merupakan 26*26.

```c
int BFS(Node start, Node goal, int size, char board[size][size]){
    Queue que; // que?
    init_queue(&que);
    int visited[size][size];
    for(int i = 0; i<size; i++){
        for(int j = 0; j<size; j++){
            visited[i][j] = 0;
        }
    }
    // Up down left right
    int dir_row[] = {-1,1,0,0};
    int dir_col[] = {0,0,-1,1};
    enqueue(&que, start);
    visited[start.row][start.col] = 1;
    while(!is_queue_empty(&que)){
        Node current = dequeue(&que);
```

```
        if(current.row == goal.row && current.col == goal.col){
            return 1;
        }
        for(int i = 0; i<4; i++){
            int new_row = current.row + dir_row[i];
            int new_col = current.col + dir_col[i];
             if( new_row>=0 && new_row<size && new_col>=0 && new_col<size
&&          board[new_row][new_col]==board[start.row][start.col]          &&
!visited[new_row][new_col]){
                visited[new_row][new_col] = 1;
                enqueue(&que,(Node){new_row,new_col});
            }
        }
    }
    return 0;
}
```

Penerapan *BFS* dilakukan seperti *BFS* pada umumnya. Keunikan untuk proyek ini hanya ada pada pengecekan *node* yang valid untuk traversal di akhir di mana ada pemeriksaan apakah warna dari *node* selanjutnya sama atau tidak. Dan apabila tidak maka tidak bisa traversal ke *node* tersebut.

PENYAMBUNGAN C DAN PYTHON

Semua penyambungan C ke Python dilakukan menggunakan *library ctypes* dari Python. Untuk kasus ini layer C di-*compile* menjadi *shared library (.SO file)* sehingga bisa diakses oleh Python.

```
self.lib = ctypes.CDLL('./queens_logic.so')
self.lib.solve_queens.argtypes = [
    ctypes.c_int, # int size
    np.ctypeslib.ndpointer(dtype=np.int8, ndim=1, flags='C_CONTIGUOUS'), #
```

```
char board[size][size]

    np.ctypeslib.ndpointer(dtype=np.int32, ndim=1, flags='C_CONTIGUOUS'),
# int solution[size][size]

    ctypes.c_int, # int freq

    CALLBACK_TYPE, # CallbackFunc cb

    ITER_COUNT_TYPE, # IterCountFunc itcountfun

    ctypes.c_int # int isPrune

]

self.lib.solve_queens.restype = ctypes.c_int
```

Kode di atas menunjukkan bagaimana penggunaan *ctypes* untuk mengambil *shared library* dan memformat fungsi dalam *shared library* itu agar bisa digunakan oleh Python. Tipe argumen yang didefinisikan di atas persis sama dengan tipe argumen pada layer C:

```
int solve_queens(int size, char board[size][size], int
solution[size][size], int freq, CallbackFunc cb, IterCountFunc itcountfun,
int isPrune)
```

Untuk CALLBACK_TYPE dan ITER_COUNT_TYPE adalah tipe fungsi yang akan diberi ke layer C agar C bisa memanggil fungsi yang didefinisikan di layer Python. Dengan ini sambungan komunikasi menjadi dua arah, di mana Python dapat memanggil fungsi C dan C dapat memanggil fungsi Python.

```
CALLBACK_TYPE = ctypes.CFUNCTYPE(ctypes.c_int, ctypes.c_int,
ctypes.POINTER(ctypes.c_int))

ITER_COUNT_TYPE = ctypes.CFUNCTYPE(None, ctypes.c_int)
```

Kutipan di atas adalah definisi tipe fungsi yang akan diberikan pada layer C.

```
// Callback type: receives size and a pointer to the flat solution array.
So that C can communicate to Python for liveupdates
```

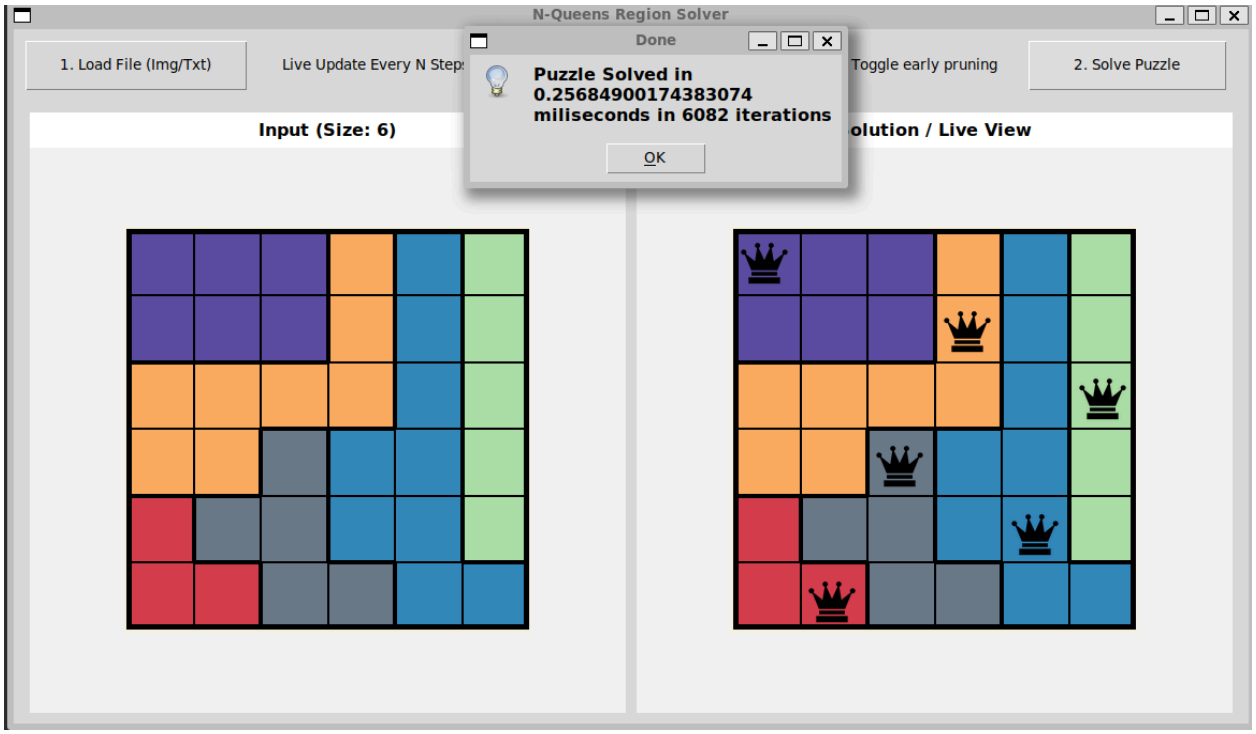
```
typedef int (*CallbackFunc)(int size, int* solution_flat);

// IterCount type: basically just reports the iter count to python since C
cannot return 2 values in one function

typedef void (*IterCountFunc)(int itercount);
```

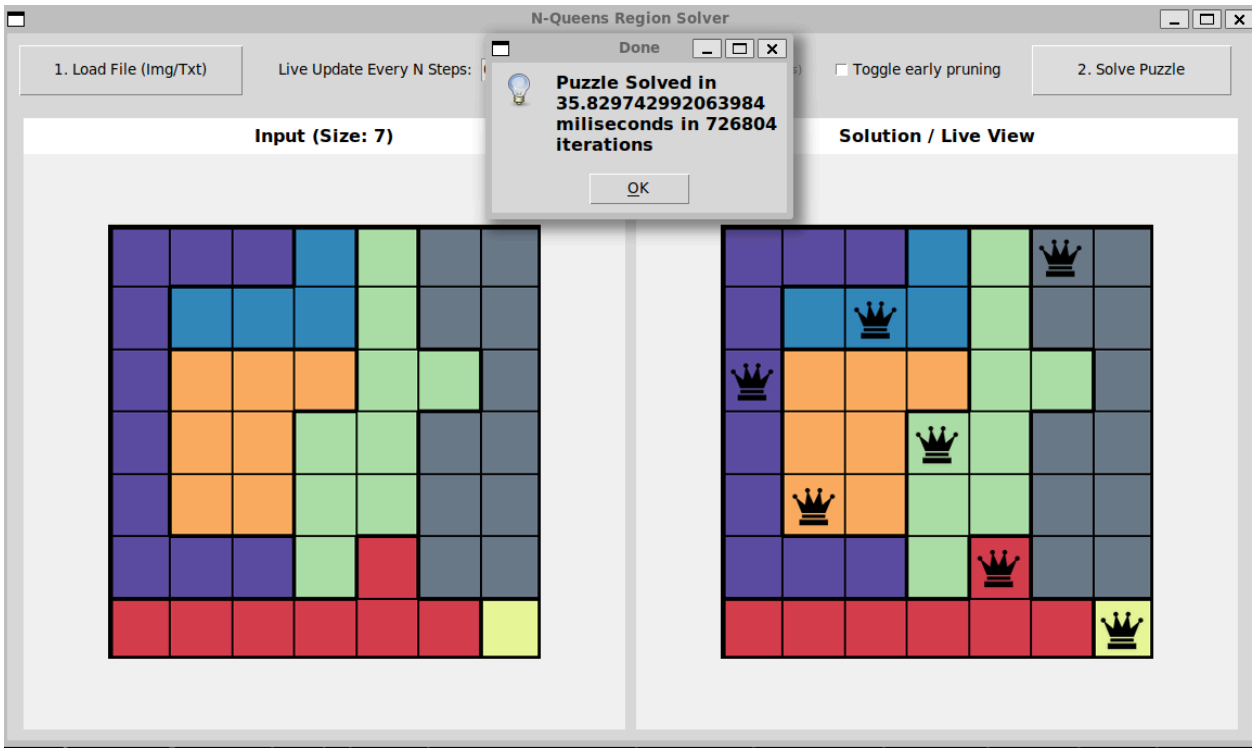
Kutipan di atas menunjukkan definisi tipe yang sama namun definisi dilakukan dalam bahasa C.

TANGKAPAN LAYAR



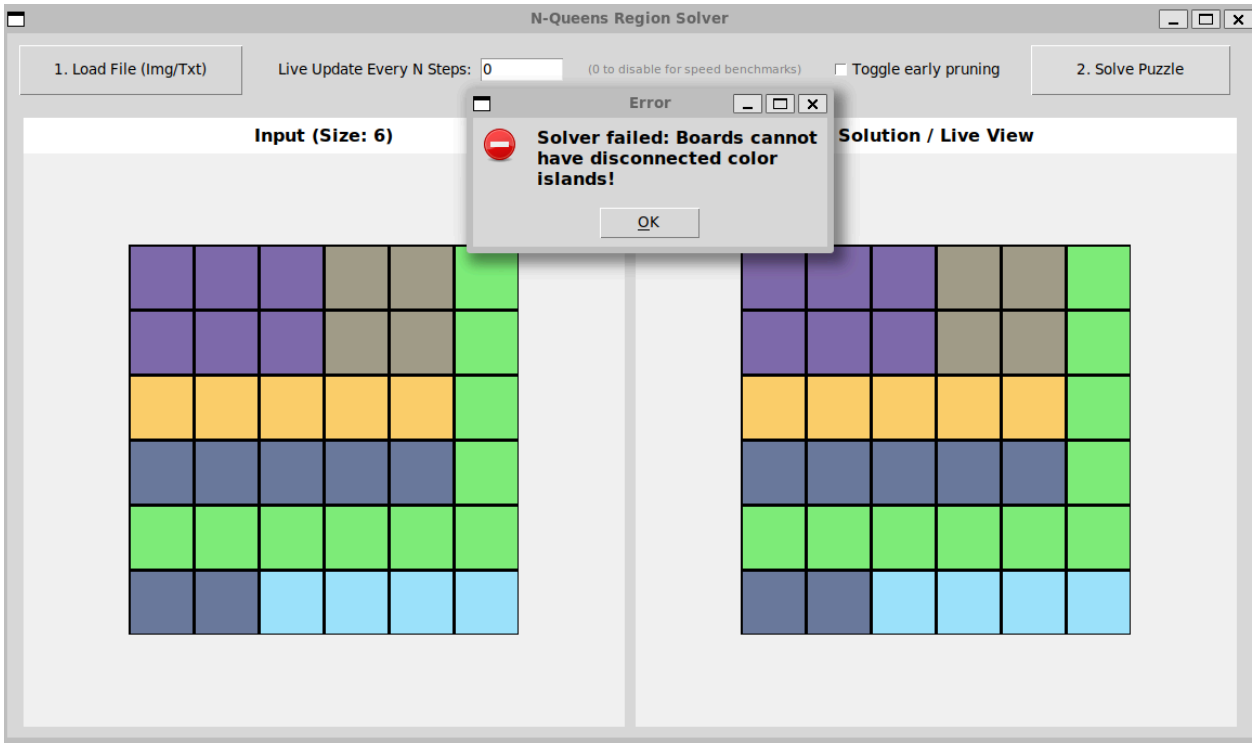

N-Queens Region Solver
Done
Puzzle Solved in 1.0410600079922006 miliseconds in 42096 iterations
OK
1. Load File (Img/Txt)
Live Update Every N Steps:
Toggle early pruning
2. Solve Puzzle
Input (Size: 9)
Solution / Live View


Done
Puzzle Solved in 2.8462339978432283 miliseconds in 63799 iterations
OK
1. Load File (Img/Txt)
Live Update Every N Steps:
Toggle early pruning
2. Solve Puzzle
Input (Size: 7)
Solution / Live View


N-Queens Region Solver
1. Load File (Img/Txt)
Live Update Every N Steps:
Toggle early pruning
2. Solve Puzzle
Input (Size: 7)
Solution / Live View
Done
Puzzle Solved in 35.829742992063984 miliseconds in 726804 iterations
OK


N-Queens Region Solver
1. Load File (Img/Txt)
Live Update Every N Steps:
Toggle early pruning
2. Solve Puzzle
Input (Size: 11)
Solution / Live View
Done
Puzzle Solved in 46.722783998120576 miliseconds in 2408215 iterations
OK


N-Queens Region Solver
1. Load File (Img/Txt)
Live Update Every N
Toggle early pruning
2. Solve Puzzle
Input (Size: 6)
Solution / Live View
Done
Puzzle Solved in 0.2390750014455989 miliseconds in 5122 iterations
OK


N-Queens Region Solver
1. Load File (Img/Txt)
Live Update Every N Steps: 0
(0 to disable for speed benchmarks)
Toggle early pruning
2. Solve Puzzle
Input (Size: 6)
Solution / Live View
Error
Solver failed: Boards cannot have disconnected color islands!
OK

PRANALA

https://github.com/billie-bytes/Tucil1_13524024

PERNYATAAN INTEGRITAS

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

Billie Bhaskara Wibawa

13524024

PESAN CINTA

Urm, QNA-nya sedikit maut, izin. Well mungkin saran saya, dalam spesifikasi definisi dari *brute force* murni itu lebih dijelaskan. Sedikit banyak diskusi dan revisi yang harus dilakukan karena definisinya ambigu. Oleh karena itu saya menerapkan toggle untuk early-prune, serta BFS untuk memeriksa adanya color islands terpisah atau tidak. Memang benar ini tidak bagian dari spek, tetapi dengan segala hormat saya minta untuk pertimbangkan menjadi nilai tambahan /\ (hand clasped emoji).

Also, yes I used a weird setup (C + Python) and also added docker just because I can :p

Goodbye para asisten. This tugas was kinda fun tbh. Idk if im gonna give this much effort on further tucils though T_T. And also semangat yak kerjanya.

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	✓	

queens_logic.c

```
#define NULL ((void*)0)

#define MAX_POSSIBLE_SQUARE 676 //Max numbers of color for 26*26 board
(kan batas max alfabet 26 yak)
```

```c
// Callback type: receives size and a pointer to the flat solution array.
So that C can communicate to Python for liveupdates

typedef int (*CallbackFunc)(int size, int* solution_flat);

// IterCount type: basically just reports the iter count to python since C
cannot return 2 values in one function

typedef void (*IterCountFunc)(int itercount);


typedef struct {

    int row;

    int col;

} Node; // Node for BFS representing {row, col}

typedef struct Queue {

     Node arr[MAX_POSSIBLE_SQUARE];

     int head;

     int tail;

} Queue;


// Globals mainly for the Python layer

static int g_iteration_count = 0;

static int g_update_freq = 0;

static CallbackFunc interrupt_func = NULL;
```

```
static void init_queue(Queue* q){
    q->head = 0;
    q->tail = -1;
}

static int is_queue_empty(Queue* q){
    return q->head > q->tail;
}

static void enqueue(Queue* q, Node node){
    q->tail++;
    q->arr[q->tail] = node;
}

static Node dequeue(Queue* q){
    if(is_queue_empty(q)){
        Node empty = {-1, -1};
        return empty;
    }

    Node temp = q->arr[q->head];
    q->head++;
```

```
    if (q->head > q->tail) {
        q->head = 0;
        q->tail = -1;
    }

    return temp;
}



/**
 * @brief BFS search
 * @return 1 if reachable, 0 if not
 */
int BFS(Node start, Node goal, int size, char board[size][size]){
    Queue que; // que?
    init_queue(&que);
    int visited[size][size];
    for(int i = 0; i<size; i++){
        for(int j = 0; j<size; j++){
            visited[i][j] = 0;
        }
    }

    // Up down left right
```

```
    int dir_row[] = {-1,1,0,0};
    int dir_col[] = {0,0,-1,1};


    enqueue(&que, start);
    visited[start.row][start.col] = 1;


    while(!is_queue_empty(&que)){
        Node current = dequeue(&que);
        if(current.row == goal.row && current.col == goal.col){
            return 1;
        }
        for(int i = 0; i<4; i++){
            int new_row = current.row + dir_row[i];
            int new_col = current.col + dir_col[i];



            if( new_row>=0 && new_row<size && new_col>=0 && new_col<size
&& board[new_row][new_col]==board[start.row][start.col] &&
!visited[new_row][new_col]){
                visited[new_row][new_col] = 1;
                enqueue(&que,(Node){new_row,new_col});
            }


        }
```

```
    }


    return 0;
}


/**
 * @brief Check if there is any disconnected color islands using BFS :/
 * @return 1 If there is no disconnected islands, 0 if yes. Oh, also -1 if jumlah color != row or col
 */
static int check_board_islands(int size, char board[size][size]){
    int is_region_checked[256]; // Same as region occuped on solve_queen() but specialized for this
    int color_num = 0;
    unsigned char current_region = '\0';

    for(int i = 0; i<256; i++){ // Memset as usual
        is_region_checked[i] = 0;
    }



    // Looks horrendous with 4 for-loops but will actually be pretty optimized since it'll skip
    // color islands that are already checked
    for(int i = 0; i<size; i++){
```

```
        for(int j = 0; j<size; j++){
            if(!is_region_checked[(unsigned char)board[i][j]]){
                color_num++;
                current_region = (unsigned char)board[i][j];



                for(int k = i; k<size; k++){
                    int l = (k==i) ? (j+1):0;


                    for(l; l<size; l++){
                        if(board[k][l]==current_region){
                            Node start = {i,j};
                            Node goal = {k,l};
                            int result =BFS(start, goal, size, board);
                            if(!result) return 0; // Found a disconnected
color island
                        }
                    }
                }


                is_region_checked[current_region] = 1;


            }
        }
```

```
    }
    if(color_num!=size){
        return -1;
    }
    return 1;
}

static int is_safe(int i, int j, int size, int solution[size][size], char
board[size][size], int occupied_region[256]){
    /* == Row-col check == */
    for(int _i = 0; _i<size; _i++){
        if(solution[_i][j]==1){
            return 0;
        }
    }
    for(int _j = 0; _j<size; _j++){
        if(solution[i][_j]==1){
            return 0;
        }
    }

    /* == Adjacency checks == */
    int ulc, urc, blc, brc; // Upper-left-corner, upper-right-corner,
etc..
```

```
    ulc = (j>0) && (i>0);
    urc = (j<size-1) && (i>0);
    blc = (j>0) && (i<size-1);
    brc = (j<size-1) && (i<size-1);

    if(ulc && solution[i-1][j-1]) return 0;
    if(urc && solution[i-1][j+1]) return 0;
    if(blc && solution[i+1][j-1]) return 0;
    if(brc && solution[i+1][j+1]) return 0;

    /* == Region check == */
    if(occupied_region[(unsigned char)board[i][j]]==1) return 0;

    return 1;
}



/**
 * @brief Recursion WITH early pruning
 */
static int solve_queen_recursion(int row, int col, int size, int
solution[size][size], char board[size][size], int occupied_region[256]){

    /* == LIVE UPDATE LOGIC == */
```

```
    g_iteration_count++;
    if (interrupt_func != NULL && g_update_freq > 0) {
        if (g_iteration_count % g_update_freq == 0) {
            // Cast 2D array to flat pointer for Python
            if(!interrupt_func(size, (int*)solution)){
                return -1; // To stop the recursion dead in its tracks and return a special code (-1) for error in python
            }
        }
    }

    int result;

    // Base cases (I didnt add a check for queen num == region num but it already works just fine I thunk)
    if(row==size) return 1;
    else if (col==size) return 0;

    if(is_safe(row,col,size,solution,board,occupied_region)){
        occupied_region[(unsigned char)board[row][col]] = 1;
        solution[row][col] = 1;
        result = solve_queen_recursion(row+1, 0, size, solution, board, occupied_region);
    }
    else return solve_queen_recursion(row, col+1, size, solution, board,
```

```
occupied_region);


    if(result==1){
        return 1;
    }
    else if(result == -1){
        return -1;
    }
    else{
        // Traceback and cleaning up the footprints (solution matrix and
occupied regions)
        solution[row][col] = 0;
        occupied_region[(unsigned char)board[row][col]] = 0;
        return solve_queen_recursion(row, col+1, size, solution, board,
occupied_region);
    }
}


// So um, early pruning is not allowed... oh well here goes nothing


static int is_safe_check_but_a_bit_different(int row, int col, int size,
int solution[size][size], char board[size][size]) {


    /* == Row col check ==*/
    for (int k = 0; k < size; k++) {
```

```
        if (k != row && solution[k][col] == 1) return 0;
        if (k != col && solution[row][k] == 1) return 0;
    }

    /* == Region Check == */
    char current_region = board[row][col];
    for (int r = 0; r < size; r++) {
        for (int c = 0; c < size; c++) {
            if (r == row && c == col) continue; // Skip self
            if (solution[r][c] == 1 && board[r][c] == current_region)
return 0;
        }
    }

    /* == Adjacency checks == */
    int ulc, urc, blc, brc; // Upper-left-corner, upper-right-corner,
etc..
    ulc = (col>0) && (row>0);
    urc = (col<size-1) && (row>0);
    blc = (col>0) && (row<size-1);
    brc = (col<size-1) && (row<size-1);

    if(ulc && solution[row-1][col-1]) return 0;
    if(urc && solution[row-1][col+1]) return 0;
```

```
    if(blc && solution[row+1][col-1]) return 0;

    if(brc && solution[row+1][col+1]) return 0;



    return 1;

}



static int is_whole_board_valid(int size, int solution[size][size], char
board[size][size]) {

    for(int r = 0; r < size; r++) {

        for(int c = 0; c < size; c++) {

            if(solution[r][c] == 1) {

                // If any queen is unsafe, the whole board is invalid

                if (!is_safe_check_but_a_bit_different(r, c, size,
solution, board)) return 0;

            }

        }

    }

    return 1;

}



/**

 * @brief Recursion without early pruning. Generates full board, then
checks validity.

 */

static int solve_slow_recursion(int row, int size, int
```

```
solution[size][size], char board[size][size]){

    /* == LIVE UPDATE LOGIC == */

    g_iteration_count++;

    if (interrupt_func != NULL && g_update_freq > 0) {

        if (g_iteration_count % g_update_freq == 0) {

            if(!interrupt_func(size, (int*)solution)){

                return -1;

            }

        }

    }


    // Base Case

    if (row == size) {

        if (is_whole_board_valid(size, solution, board)) {

            return 1;

        }

        return 0;

    }


    int result;




    for (int col = 0; col < size; col++){

        solution[row][col] = 1;
```

```
        result = solve_slow_recursion(row + 1, size, solution, board);

        if (result == 1) {
            return 1;
        }
        else if (result == -1){
            return -1;
        }

        // Backtrack
        solution[row][col] = 0;
    }

    return 0;
}



int solve_queens(int size, char board[size][size], int
solution[size][size], int freq, CallbackFunc cb, IterCountFunc itcountfun,
int isPrune){
    int occupied_region[256];

    // Setup globals
```

```
    g_iteration_count = 0;

    g_update_freq = freq;

    interrupt_func = cb;



    // Memsets

    for(int i = 0; i<size; i++){for(int j = 0; j<size; j++){solution[i][j]
= 0;}}

    for(int i = 0; i<256; i++){occupied_region[i] = 0;}



    int board_validity = check_board_islands(size,board);

    if(board_validity==0){

        return -2; // Board islands disconnected

    }

    else if(board_validity==-1){

        return -3; // Board colors amount != row or col

    }



    int result;

    if(isPrune){

        result = solve_queen_recursion(0, 0, size, solution, board,
occupied_region);

    }

    else{

        result = solve_slow_recursion(0, size, solution, board);
```

```
    }


    itcountfun(g_iteration_count);
    return result;
}
```

image_io.py

```
import cv2
import numpy as np
import random
from PIL import Image, ImageDraw
import os


def are_colors_similar(c1, c2, threshold=20):
    """
    Without this the color parsing is way less accurate
    """
    r1, g1, b1 = c1
    r2, g2, b2 = c2
    dist = np.sqrt((r1 - r2)**2 + (g1 - g2)**2 + (b1 - b2)**2)
    return dist < threshold


def process_board_input(image_path, grid_size=8):
    img = cv2.imread(image_path)
```

```
    if img is None:
        raise ValueError("Could not load image.")

    target_dim = grid_size * 50
    img_resized = cv2.resize(img, (target_dim, target_dim),
interpolation=cv2.INTER_AREA)

    cell_width = target_dim // grid_size
    cell_height = target_dim // grid_size

    total_cells = grid_size * grid_size
    region_matrix = ['X'] * total_cells

    known_colors = {}
    region_counter = 1

    sample_ratio = 0.2
    half_sample_w = max(1, int((cell_width * sample_ratio) / 2))
    half_sample_h = max(1, int((cell_height * sample_ratio) / 2))

    for row in range(grid_size):
        for col in range(grid_size):
            flat_index = (row * grid_size) + col
            center_x = (col * cell_width) + (cell_width // 2)
```

```
            center_y = (row * cell_height) + (cell_height // 2)


            patch = img_resized[
                center_y - half_sample_h : center_y + half_sample_h,
                center_x - half_sample_w : center_x + half_sample_w
            ]


            if patch.size == 0:
                pixel_color = tuple(img_resized[center_y, center_x])
            else:
                median_color = np.median(patch.reshape(-1, 3), axis=0)
                pixel_color = tuple(median_color.astype(int))


            found_existing = False
            for existing_color, region_char in known_colors.items():
                if are_colors_similar(pixel_color, existing_color):
                    region_matrix[flat_index] = region_char
                    found_existing = True
                    break


            if not found_existing:
                new_char = chr(region_counter + 64)
                known_colors[pixel_color] = new_char
                region_matrix[flat_index] = new_char
```

```
                region_counter += 1


    return region_matrix


def create_board_from_text(text_content, grid_size,
output_path="temp_text_board.png"):


    clean_text = "".join(text_content.split())
    if len(clean_text) != grid_size * grid_size:
        raise ValueError(f"Text length ({len(clean_text)}) does not match
grid size")


    cell_size = 60
    img_size = grid_size * cell_size
    img = Image.new("RGB", (img_size, img_size), "white")
    draw = ImageDraw.Draw(img)


    # Randomize colors
    random.seed(42)
    color_map = {}


    for r in range(grid_size):
        for c in range(grid_size):
            char = clean_text[r * grid_size + c]
```

```
            if char not in color_map:


                color_map[char] = (

                    random.randint(100, 255),

                    random.randint(100, 255),

                    random.randint(100, 255)

                )


            x0 = c * cell_size

            y0 = r * cell_size

            x1 = x0 + cell_size

            y1 = y0 + cell_size


            # Draw cell background

            draw.rectangle([x0, y0, x1, y1], fill=color_map[char],
outline="black", width=2)




    img.save(output_path)

    return output_path, list(clean_text)


def generate_board_output(original_image_path, solution_matrix,
output_path="output/solution.png", output_size=8):

    base_image = Image.open(original_image_path).convert("RGBA")
```

```
    width, height = base_image.size


    cell_w = width / output_size
    cell_h = height / output_size


    try:
        queen_icon = Image.open("queen_asset.png").convert("RGBA")
        icon_size = int(min(cell_w, cell_h) * 0.8)
        queen_icon = queen_icon.resize((icon_size, icon_size),
Image.Resampling.LANCZOS)
    except FileNotFoundError:
        queen_icon = None


    draw = ImageDraw.Draw(base_image)


    for row in range(output_size):
        for col in range(output_size):
            flat_index = (row * output_size) + col
            if solution_matrix[flat_index] == 1:
                x_pos = int((col * cell_w) + (cell_w / 2))
                y_pos = int((row * cell_h) + (cell_h / 2))


                if queen_icon:
                    offset_x = int(x_pos - (queen_icon.width // 2))
```

```
                    offset_y = int(y_pos - (queen_icon.height // 2))
                    base_image.alpha_composite(queen_icon, (offset_x,
offset_y))
                else:
                    r = min(cell_w, cell_h) * 0.3
                    draw.ellipse(
                        (x_pos - r, y_pos - r, x_pos + r, y_pos + r),
                        fill=(255, 0, 0, 255), outline="black"
                    )
    os.makedirs("output", exist_ok=True)
    base_image.save(output_path)
    return output_path
```

queens_interface.py

```
import tkinter as tk
from tkinter import filedialog, messagebox, simpledialog
from PIL import Image, ImageTk
import ctypes
import numpy as np
import time


from image_io import process_board_input, generate_board_output,
create_board_from_text
```

```
# Callback type: receives size and a pointer to the flat solution array.
So that C can communicate to Python for liveupdates

CALLBACK_TYPE = ctypes.CFUNCTYPE(ctypes.c_int, ctypes.c_int,
ctypes.POINTER(ctypes.c_int))

ITER_COUNT_TYPE = ctypes.CFUNCTYPE(None, ctypes.c_int)


class QueensSolverApp:

    def __init__(self, root):

        self.root = root

        self.root.title("N-Queens Region Solver")

        self.root.geometry("1150x650")


        # Load the C stuff

        try:

            self.lib = ctypes.CDLL('./queens_logic.so')

            self.lib.solve_queens.argtypes = [

                ctypes.c_int, # int size

                np.ctypeslib.ndpointer(dtype=np.int8, ndim=1,
flags='C_CONTIGUOUS'), # char board[size][size]

                np.ctypeslib.ndpointer(dtype=np.int32, ndim=1,
flags='C_CONTIGUOUS'), # int solution[size][size]

                ctypes.c_int, # int freq

                CALLBACK_TYPE, # CallbackFunc cb

                ITER_COUNT_TYPE, # IterCountFunc itcountfun

                ctypes.c_int # int isPrune
```

```
            ]
            self.lib.solve_queens.restype = ctypes.c_int
        except OSError:
            messagebox.showerror("Error", "Could not load .SO file.")
            self.root.destroy()
            return

        self.grid_size = 8
        self.current_image_path = None
        self.region_data = None
        self.step_counter = 0
        self.early_prune = tk.BooleanVar()
        self.early_prune.set(False)

        self.ui_stuff()

    def ui_stuff(self):


        control_frame = tk.Frame(self.root, pady=10)
        control_frame.pack(side=tk.TOP, fill=tk.X)

        self.lbl_loading = tk.Label(control_frame, text="Solving...",
fg="red", font=("Arial", 12, "bold"))
```

```
        btn_load = tk.Button(control_frame, text="1. Load File (Img/Txt)",
command=self.load_file, width=20, height=2)
        btn_load.pack(side=tk.LEFT, padx=10)

        # Step N Input
        lbl_step = tk.Label(control_frame, text="Live Update Every N
Steps:", font=("Arial", 10))
        lbl_step.pack(side=tk.LEFT, padx=(20, 5))
        self.entry_step = tk.Entry(control_frame, width=8, font=("Arial",
10))
        self.entry_step.insert(0, "0") # Default 0 = disabled
        self.entry_step.pack(side=tk.LEFT, padx=(0, 20))
        lbl_note = tk.Label(control_frame, text="(0 to disable for speed
benchmarks)", fg="gray", font=("Arial", 8))
        lbl_note.pack(side=tk.LEFT)

        check_prune = tk.Checkbutton(control_frame, text="Toggle early
pruning", font=("Arial",10), variable=self.early_prune)
        check_prune.pack(side=tk.LEFT, padx=(20,5))

        self.btn_solve = tk.Button(control_frame, text="2. Solve Puzzle",
command=self.solve_puzzle, width=20, height=2, bg="#dddddd")
        self.btn_solve.pack(side=tk.RIGHT, padx=20)
```

```
        # Main Frame
        image_frame = tk.Frame(self.root)
        image_frame.pack(side=tk.TOP, expand=True, fill=tk.BOTH, padx=10,
pady=10)


        # Left Panel
        self.panel_left = tk.Frame(image_frame, bg="white", width=400,
height=400)
        self.panel_left.pack(side=tk.LEFT, expand=True, fill=tk.BOTH,
padx=5)
        self.label_input_title = tk.Label(self.panel_left, text="Input
Board", bg="white", font=("Arial", 12, "bold"))
        self.label_input_title.pack(side=tk.TOP, pady=5)
        self.label_input_img = tk.Label(self.panel_left, bg="#f0f0f0",
text="No File Loaded")
        self.label_input_img.pack(expand=True, fill=tk.BOTH)


        # Right Panel
        self.panel_right = tk.Frame(image_frame, bg="white", width=400,
height=400)
        self.panel_right.pack(side=tk.RIGHT, expand=True, fill=tk.BOTH,
padx=5)
        self.label_output_title = tk.Label(self.panel_right,
text="Solution / Live View", bg="white", font=("Arial", 12, "bold"))
        self.label_output_title.pack(side=tk.TOP, pady=5)
        self.label_output_img = tk.Label(self.panel_right, bg="#f0f0f0")
```

```
        self.label_output_img.pack(expand=True, fill=tk.BOTH)


    def load_file(self):
        file_path = filedialog.askopenfilename(
            title="Select Board File",
            filetypes=[
                ("All Supported", "*.png *.jpg *.jpeg *.txt"),
                ("Images", "*.png *.jpg *.jpeg"),
                ("Text Files", "*.txt")
            ]
        )
        if not file_path:
            return


        try:
            if file_path.lower().endswith(".txt"):
                with open(file_path, 'r') as f:
                    content = f.read()


                # Dimension calculations
                row_length = content.find('\n')
                string_length = len("".join(content.split()))
                if(string_length!=(row_length**2)):
                    raise ValueError("Each row must have the same number
```

```
of characters!")
                for i in range(string_length):
                    if((i%(row_length+1)==row_length) and
content[i]!='\n'):
                        raise ValueError("Each row must have the same
number of characters!")
                self.grid_size = row_length


                # Create image from txt
                img_path, region_list = create_board_from_text(content,
self.grid_size)
                self.current_image_path = img_path
                self.region_data = region_list
                print(f"Text processed. Image at {img_path}")
                self.display_image(self.current_image_path,
self.label_input_img)


            else:
                # Handle Image Input
                self.current_image_path = file_path
                self.display_image(self.current_image_path,
self.label_input_img)
                size_val = simpledialog.askinteger("Grid Size", "What is
the size of the board? (e.g. 8)", minvalue=4, maxvalue=50)
                if not size_val:
                    return
```

```
                self.grid_size = size_val
                self.region_data = process_board_input(file_path,
self.grid_size)
                print(f"Image processed for size {self.grid_size}.")

            # Display the input image
            self.label_input_title.config(text=f"Input (Size:
{self.grid_size})")

        except Exception as e:
            messagebox.showerror("Error", f"Failed to process file: {e}")

    def c_interrupt(self, size, solution_ptr):
        """
        Called from C library during recursion for liveupdates
        """
        try:
            solution_arr = np.ctypeslib.as_array(solution_ptr,
shape=(size*size,))
            self.step_counter += 1
            file_path = f"output/process_step_{self.step_counter}.png"

            generate_board_output(self.current_image_path, solution_arr,
file_path, size)
```

```
            # Update UI (Must update idletasks to refresh UI during
blocking C call)
            self.display_image(file_path, self.label_output_img)
            self.root.update()
            return 1
        except Exception as e:
            print(f"Callback Error: {e}")
            return 0


    def c_iter_count(self, itercount):
        """
        Also called from C to report the iteration count
        """
        self.step_counter = itercount


    def save_text_solution(self, solution_array):
        """
        Saves the solution to a text file where '#' represents a queen.
        """
        try:
            with open("output/final_solution.txt", "w") as f:
                for r in range(self.grid_size):
                    row_str = ""
                    for c in range(self.grid_size):
```

```
                    idx = r * self.grid_size + c
                    if solution_array[idx] == 1:
                        row_str += "#"
                    else:
                        row_str += self.region_data[idx]
                f.write(row_str + "\n")
        print("Text solution saved to output/final_solution.txt")
    except Exception as e:
        print(f"Failed to save text solution: {e}")


def solve_puzzle(self):
    if not self.region_data:
        messagebox.showwarning("Warning", "Please load a file first.")
        return


    try:
        step_n = int(self.entry_step.get())
    except ValueError:
        step_n = 0


    self.step_counter = 0


    try:
        board_array = np.array([ord(c) for c in self.region_data],
```

```
dtype=np.int8)

            solution_array = np.zeros(self.grid_size * self.grid_size,
dtype=np.int32)



            # To pass inside the C component

            c_func = CALLBACK_TYPE(self.c_interrupt)

            c_iter_func = ITER_COUNT_TYPE(self.c_iter_count)



            self.lbl_loading.pack(side=tk.LEFT, padx=10)

            self.root.update()



            # Sikat

            result = None

            start_time = time.perf_counter()

            if self.early_prune.get():

                result = self.lib.solve_queens(self.grid_size,
board_array, solution_array, step_n, c_func, c_iter_func, 1)

            else:

                result = self.lib.solve_queens(self.grid_size,
board_array, solution_array, step_n, c_func, c_iter_func, 0)



            time_elapsed = time.perf_counter() - start_time



            # Final Output after liveupdates

            output_path = "output/final_solution.png"
```

```
            generate_board_output(self.current_image_path, solution_array,
output_path, self.grid_size)

            self.display_image(output_path, self.label_output_img)

            self.save_text_solution(solution_array)



            if result == -1:

                return

            elif result == -2:

                raise ValueError("Boards cannot have disconnected color
islands!")

            elif result == -3:

                raise ValueError("Number of colors doesn't match size of
board!")

            elif result == 0:

                messagebox.showinfo("Done", f"No solutions found :(")

            else:

                messagebox.showinfo("Done", f"Puzzle Solved in
{time_elapsed*1000} miliseconds in {self.step_counter} iterations")



        except Exception as e:

            messagebox.showerror("Error", f"Solver failed: {e}")

            return

        finally:

            self.lbl_loading.pack_forget()
```

```
    def display_image(self, path, label_widget):
        img = Image.open(path)
        img.thumbnail((400, 400))
        tk_img = ImageTk.PhotoImage(img)

        label_widget.config(image=tk_img, text="")
        label_widget.image = tk_img


if __name__ == "__main__":
    root = tk.Tk()
    app = QueensSolverApp(root)
    root.mainloop()
```