

Laporan Tugas Kecil 1 IF2211 Strategi Algoritma
Semester II Tahun 2025/2026
Penyelesaian N- Queens LinkedIn



Disusun oleh:

Nama: Al Farabi
NIM: 13524086
Kelas: K02

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025**

I. Pendahuluan

A. Latar Belakang

Permainan Queens pada platform LinkedIn merupakan pengembangan dari masalah klasik N-Queens yang sering dipelajari dalam materi strategi algoritma. Pada permainan ini, pemain harus menempatkan Queen pada papan berukuran $N \times N$ dengan beberapa aturan, yaitu setiap baris dan kolom hanya boleh berisi satu Queen. Selain itu, setiap daerah warna hanya boleh berisi satu Queen, dan Queen tidak boleh saling bersebelahan termasuk secara diagonal.

Karena banyaknya aturan yang harus dipenuhi secara bersamaan, tidak semua susunan Queen dapat menjadi solusi yang valid. Seiring bertambahnya ukuran papan, jumlah kemungkinan konfigurasi juga meningkat secara signifikan sehingga proses pencarian solusi menjadi lebih menantang. Oleh karena itu, diperlukan suatu metode yang dapat mencoba berbagai kemungkinan penempatan secara sistematis.

Pada tugas ini digunakan algoritma brute force untuk mencari solusi. Pendekatan ini dilakukan dengan mencoba semua kemungkinan konfigurasi yang ada, dengan tiga strategi berbeda yang memiliki tingkat optimasi berbeda. Strategi pertama adalah pure brute force yang mencoba semua posisi untuk setiap queen. Strategi kedua menggunakan optimasi satu queen per kolom. Strategi ketiga menambahkan early pruning untuk memangkas cabang yang tidak valid sejak dini. Dengan cara ini, proses pencarian dapat diamati secara langsung, termasuk berapa banyak konfigurasi yang diperiksa dan berapa lama waktu yang dibutuhkan untuk menemukan solusi. Strategi algoritma dibuat berbeda karena ada kebingungan terkait penggunaan algoritma backtracking pada tugas ini, yang mengakibatkan pengambilan keputusan saya untuk menggunakan 3 strategi algoritma optimasi yang berbeda, dari mulai pure brute force, one queen per column, dan early pruning.

B. Rumusan Masalah

Berdasarkan latar belakang tersebut, rumusan masalah dalam tugas ini adalah:

1. Bagaimana merancang algoritma brute force untuk menemukan solusi permainan Queens yang memenuhi seluruh constraint?
2. Bagaimana memastikan bahwa konfigurasi yang dihasilkan valid terhadap aturan baris, kolom, daerah warna, dan larangan bersebelahan?
3. Bagaimana mengoptimalkan algoritma brute force dengan strategi one queen per column dan early pruning?
4. Bagaimana menghitung jumlah konfigurasi yang ditinjau selama proses pencarian?
5. Bagaimana mengukur dan membandingkan waktu eksekusi ketiga strategi algoritma?

C. Tujuan

Tujuan dari pelaksanaan tugas ini adalah :

1. Mengimplementasikan tiga strategi algoritma brute force untuk menyelesaikan permainan Queens sesuai dengan spesifikasi yang diberikan.
2. Menghasilkan solusi yang valid atau menyatakan bahwa tidak terdapat solusi.
3. Menghitung dan menampilkan jumlah konfigurasi yang ditinjau selama proses pencarian.
4. Mengukur waktu eksekusi masing-masing strategi algoritma pencarian solusi.

5. Menganalisis performa ketiga strategi algoritma terhadap ukuran papan dan kompleksitas kasus.

II. Algoritma Brute Force

A. Ide Dasar Algoritma

Pada tugas ini, permainan Queens diselesaikan menggunakan metode brute force dengan tiga strategi berbeda. Program akan mencoba berbagai kemungkinan peletakan queen sampai menemukan konfigurasi yang memenuhi semua aturan. Ketiga strategi yang diimplementasikan adalah:

1. Pure Brute Force: Mencoba semua kemungkinan posisi untuk setiap queen tanpa constraint apapun.
2. One Queen Per Column: Membatasi setiap queen pada kolom yang berbeda untuk mengurangi ruang pencarian.
3. Early Pruning: Menambahkan validasi dini sebelum melakukan rekursi untuk memangkas cabang yang tidak valid.

Jumlah Queen yang harus ditempatkan sama dengan ukuran papan N . Jika papan berukuran $N \times N$, maka harus ditempatkan tepat N queen. Program kemudian mencoba berbagai kombinasi posisi dan mengecek apakah kombinasi tersebut memenuhi aturan permainan. Proses ini terus dilakukan sampai solusi ditemukan atau semua kemungkinan sudah dicoba.

B. Pseudocode

1. Strategi 1: Pure Brute Force

```
ALGORITMA solveBruteForce(papan, indeksQueen,
mulaiDari)
{ Mencoba semua kemungkinan posisi untuk setiap queen
}

DEKLARASI
    totalSel : integer
    baris, kolom : integer

ALGORITMA
    totalSel  $\leftarrow N \times N$ 

    { Base case: semua queen sudah ditempatkan }
    if indeksQueen =  $N$  then
        tambah iterationCount

        { Visualisasi jika perlu }
        if iterationCount mod freqVisualisasi = 0 then
            tampilkan papan dan nomor iterasi
        endif

        { Validasi konfigurasi lengkap }
        if isValid( $N$ ) then
            tampilkan "SOLUSI DITEMUKAN"
```

```

        tampilkan papan
        return true
    else
        return false
    endif
endif

    { Recursive case: coba semua posisi untuk queen
saat ini }
    for idx ← 0 to totalSel - 1 do
        { Konversi indeks linear ke koordinat (baris,
kolom) }
        baris ← idx div N
        kolom ← idx mod N

        { Hapus queen dari posisi lama (jika bukan
queen pertama) }
        if indeksQueen > 0 OR idx > 0 then
            barisLama ← posisiQueens[indeksQueen][0]
            kolomLama ← posisiQueens[indeksQueen][1]
            papan[barisLama][kolomLama] ←
papanAsli[barisLama][kolomLama]
        endif

        { Tempatkan queen di posisi baru }
        papan[baris][kolom] ← "#"
        posisiQueens[indeksQueen] ← [baris, kolom]

        { Rekursi untuk queen berikutnya }
        if solveBruteForce(papan, indeksQueen + 1, 0)
then
            return true
        endif
    endfor

    { Backtracking: kembalikan queen ke posisi awal }
    barisLama ← posisiQueens[indeksQueen][0]
    kolomLama ← posisiQueens[indeksQueen][1]
    papan[barisLama][kolomLama] ←
papanAsli[barisLama][kolomLama]
    posisiQueens[indeksQueen] ← [0, 0]
    papan[0][0] ← "#"

```

```
return false
```

2. Strategi 2: One Queen Per Column

```
ALGORITMA solveQueenKolom(papan, indeksQueen)
{ Menempatkan satu queen per kolom untuk mengurangi
ruang pencarian }

DEKLARASI
    indeksKolom, baris : integer

ALGORITMA
    { Base case: semua queen sudah ditempatkan }
    if indeksQueen = N then
        tambah iterationCount

        { Visualisasi jika perlu }
        if iterationCount mod freqVisualisasi = 0 then
            tampilkan papan dan nomor iterasi
        endif

        { Validasi konfigurasi lengkap }
        if isValid(N) then
            tampilkan "SOLUSI DITEMUKAN"
            tampilkan papan
            return true
        else
            return false
        endif
    endif

    { Constraint: queen ke-i di kolom i }
    indeksKolom ← indeksQueen

    { Recursive case: coba semua baris pada kolom ini }
    for baris ← 0 to N - 1 do
        { Hapus queen dari posisi lama (jika bukan
        baris pertama) }
        if baris > 0 then
            barisLama ← posisiQueens[indeksQueen][0]
            kolomLama ← posisiQueens[indeksQueen][1]
            papan[barisLama][kolomLama] ←
papanAsli[barisLama][kolomLama]
        endif
```

```

        { Tempatkan queen di posisi baru }
        papan[baris][indeksKolom] ← "#"
        posisiQueens[indeksQueen] ← [baris,
indeksKolom]

        { Rekursi untuk queen berikutnya }
        if solveQueenKolom(papan, indeksQueen + 1)
then
            return true
        endif
    endfor

    { Backtracking: kembalikan queen ke baris 0 }
    barisLama ← posisiQueens[indeksQueen][0]
    kolomLama ← posisiQueens[indeksQueen][1]
    papan[barisLama][kolomLama] ←
papanAsli[barisLama][kolomLama]
    posisiQueens[indeksQueen] ← [0, indeksKolom]
    papan[0][indeksKolom] ← "#"

    return false

```

3. Strategi 3: Early Pruning + One Queen Per Column

```

ALGORITMA solvePruning(papan, indeksQueen)
{ Menggunakan validasi dini untuk memangkas cabang
tidak valid }

DEKLARASI
    indeksKolom, baris : integer
    aman : boolean

ALGORITMA
    { Base case: semua queen sudah ditempatkan }
    if indeksQueen = N then
        tambah iterationCount

        { Visualisasi jika perlu }
        if iterationCount mod freqVisualisasi = 0 then
            tampilkan papan dan nomor iterasi
        endif

        { Double-check validasi }
        if isValid(N) then

```

```

        tampilkan "SOLUSI DITEMUKAN"
        tampilkan papan
        return true
    else
        return false
    endif
endif

{ Constraint: queen ke-i di kolom i }
indeksKolom ← indeksQueen

{ Recursive case: coba semua baris dengan early
pruning }
for baris ← 0 to N - 1 do
    tambah iterationCount

    { Tempatkan queen sementara untuk visualisasi
}

    papan[baris][indeksKolom] ← "#"
    posisiQueens[indeksQueen] ← [baris,
indeksKolom]

    { Visualisasi untuk iterasi awal (debugging) }
    if iterationCount ≤ 50 then
        tampilkan papan dan nomor iterasi
    endif

    { EARLY PRUNING: validasi sebelum rekursi }
    aman ← isSquareSafe(baris, indeksKolom,
indeksQueen)

    { Rekursi HANYA jika posisi aman }
    if aman then
        if solvePruning(papan, indeksQueen + 1)
then
            return true
        endif
    endif
endif

{ Backtracking: hapus queen (baik aman maupun
tidak) }
papan[baris][indeksKolom] ←
papanAsli[baris][indeksKolom]

```

```

        endfor

        return false

{ Fungsi helper untuk early pruning }
FUNGSI isSquareSafe(baris, kolom, indeksQueen) →
boolean
{ Memeriksa apakah posisi aman terhadap queen yang
sudah ada }

    { Cek setiap queen yang sudah ditempatkan }
    for i ← 0 to indeksQueen - 1 do
        barisQ ← posisiQueens[i][0]
        kolomQ ← posisiQueens[i][1]

        { Cek konflik vertikal }
        if barisQ = baris then
            return false
        endif

        { Cek konflik horizontal }
        if kolomQ = kolom then
            return false
        endif

        { Cek konflik tetangga (jarak ≤ 1) }
        if isTetangga(barisQ, baris, kolomQ, kolom)
then
            return false
        endif

        { Cek konflik warna }
        if isWarnaSama(barisQ, kolomQ, baris, kolom)
then
            return false
        endif
    endfor

    return true

```

C. Langkah-langkah Algoritma

1. Strategi 1: Pure Brute Force

Program dimulai dengan meminta pengguna memasukkan nama file input, pilihan strategi, dan frekuensi visualisasi. File input kemudian dibaca dan disimpan ke dalam array dua dimensi yang merepresentasikan papan permainan. Dari pembacaan tersebut, program menentukan ukuran papan N dan menyimpan karakter asli setiap sel untuk validasi warna.

Setelah informasi dasar papan diketahui, program menginisialisasi struktur data untuk menyimpan posisi setiap queen. Semua queen dimulai dari posisi $(0,0)$ sebagai konfigurasi awal. Proses pencarian dimulai dengan mencatat waktu awal menggunakan fungsi pengukuran waktu dari library time. Selanjutnya, fungsi `solveBruteForce()` dipanggil. Di dalam fungsi ini, program akan mencoba menempatkan queen secara rekursif.

Untuk setiap queen, program melakukan iterasi melalui semua sel pada papan (dari 0 hingga N^2-1). Setiap posisi sel dicoba sebagai tempat queen saat ini. Posisi linear dikonversi menjadi pasangan koordinat (baris, kolom) menggunakan operasi pembagian dan modulo.

Base case: Jika semua N queen sudah ditempatkan (indeks queen sama dengan N), program menambahkan penghitung iterasi dan melakukan visualisasi jika diperlukan. Kemudian konfigurasi divalidasi menggunakan fungsi `isValid()`. Jika valid, solusi ditemukan dan ditampilkan. Jika tidak valid, fungsi mengembalikan false untuk backtracking.

Recursive case: Untuk setiap posisi sel yang dicoba:

- Queen dihapus dari posisi lama pada papan kerja
- Queen ditempatkan di posisi baru
- Fungsi rekursif dipanggil untuk queen berikutnya
- Jika rekursi berhasil menemukan solusi, return true

Jika tidak ada posisi yang menghasilkan solusi, queen dikembalikan ke posisi $(0,0)$ dan fungsi mengembalikan false. Kompleksitas waktu strategi ini adalah $O((N^2)^N)$ karena setiap dari N queen dapat ditempatkan di N^2 posisi.

2. Strategi 2: One Queen Per Column

Strategi ini menambahkan constraint bahwa queen ke- i harus berada di kolom i . Hal ini secara signifikan mengurangi ruang pencarian. Proses dimulai sama dengan strategi pertama hingga pemanggilan fungsi solver. Pada strategi ini, fungsi `solveQueenKolom()` dipanggil. Untuk setiap queen, program hanya melakukan iterasi melalui baris pada kolom yang ditentukan (kolom = indeks queen). Jadi queen pertama hanya mencoba baris 0 hingga $N-1$ pada kolom 0, queen kedua pada kolom 1, dan seterusnya.

Base case: Sama seperti strategi pertama, ketika semua queen sudah ditempatkan, konfigurasi divalidasi.

Recursive case: Untuk setiap baris pada kolom yang ditentukan:

- Queen dihapus dari posisi lama (jika ini bukan iterasi pertama)
- Queen ditempatkan di (baris, kolom_indeks_queen)
- Fungsi rekursif dipanggil untuk queen berikutnya
- Jika berhasil, return true

Backtracking: Queen dikembalikan ke $(0, \text{kolom_indeks_queen})$.

Kompleksitas waktu strategi ini adalah $O(N^N)$ karena setiap dari N queen dapat ditempatkan di N posisi (hanya baris yang bervariasi).

3. Strategi 3: Early Pruning + One Queen Per Column

Strategi ini mengombinasikan constraint kolom dengan validasi dini untuk memangkas cabang yang tidak valid sebelum melakukan rekursi lebih lanjut. Fungsi `solvePruning()` memiliki struktur yang mirip dengan strategi kedua, namun dengan penambahan validasi dini.

Untuk setiap posisi yang dicoba:

- Queen ditempatkan sementara di posisi (baris, kolom_indeks_queen)
- Counter iterasi ditambahkan
- Visualisasi dilakukan jika diperlukan (untuk 50 iterasi pertama)
- Validasi dini: Fungsi `isSquareSafe()` dipanggil untuk memeriksa apakah posisi ini aman terhadap queen yang sudah ditempatkan sebelumnya
- Conditional recursion: Hanya jika posisi aman, rekursi dilakukan untuk queen berikutnya
- Backtracking: Queen dihapus dari posisi tersebut (baik aman maupun tidak)

Fungsi `isSquareSafe()` melakukan empat jenis pemeriksaan terhadap semua queen yang sudah ditempatkan:

1. Konflik vertikal (baris sama)
2. Konflik horizontal (kolom sama)
3. Konflik tetangga (jarak ≤ 1 dalam 8 arah)
4. Konflik warna (karakter papan asli sama)

Jika terdapat konflik apapun, fungsi mengembalikan false dan posisi tersebut di-skip tanpa rekursi.

Strategi ini memiliki kompleksitas terbaik karena banyak cabang yang dipangkas sejak dini. Dalam kasus terbaik, kompleksitasnya mendekati $O(N!)$ karena setiap queen memiliki pilihan yang semakin sedikit seiring bertambahnya constraint.

Setelah proses pencarian selesai (baik menemukan solusi atau tidak), program mencatat waktu akhir dan menghitung selisih waktu dalam satuan milidetik. Hasil kemudian ditampilkan di terminal, termasuk papan solusi jika ada, waktu pencarian, dan jumlah konfigurasi yang telah diperiksa. Jika solusi ditemukan, pengguna diberi pilihan untuk menyimpan hasil ke dalam file dengan nama `solusi_<nama_input>.txt`.

D. Kompleksitas Waktu

1. Strategi 1: Pure Brute Force

Kompleksitas waktu strategi ini adalah $O((N^2)^N)$. Hal ini karena:

- Terdapat N queen yang harus ditempatkan
- Setiap queen dapat ditempatkan di N^2 posisi (semua sel papan)
- Total kombinasi yang harus diperiksa adalah $(N^2)^N$

Untuk setiap konfigurasi yang diperiksa, validasi memerlukan waktu $O(N^2)$ karena harus membandingkan setiap pasangan queen. Dengan demikian, kompleksitas total adalah:

$$O((N^2)^N \times N^2)$$

Untuk papan 9x9, ini berarti mencoba $81^9 \approx 1.97 \times 10^{17}$ kombinasi, yang praktis tidak mungkin diselesaikan dalam waktu wajar.

2. Strategi 2: One Queen Per Column

Dengan constraint satu queen per kolom, kompleksitas waktu menjadi $O(N^N)$. Hal ini karena:

- Terdapat N queen yang harus ditempatkan
- Setiap queen hanya dapat ditempatkan di N posisi (baris pada kolomnya)
- Total kombinasi yang harus diperiksa adalah N^N

Untuk setiap konfigurasi, validasi tetap memerlukan waktu $O(N^2)$.

Kompleksitas total:

$$O(N^N \times N^2)$$

Untuk papan 9×9 , ini berarti mencoba $9^9 \approx 387$ juta kombinasi, yang jauh lebih feasible namun masih lambat.

3. Strategi 3: Early Pruning

Strategi ini memiliki kompleksitas terbaik karena banyak cabang yang dipangkas. Dalam kasus terburuk, kompleksitasnya masih $O(N^N)$, namun dalam praktiknya mendekati $O(N!)$ karena:

- Setiap queen yang ditempatkan mengurangi pilihan yang valid untuk queen berikutnya
- Banyak posisi yang di-skip tanpa rekursi karena tidak lolos validasi dini

Validasi dini menggunakan `isSquareSafe()` memerlukan waktu $O(N)$ karena hanya memeriksa queen yang sudah ditempatkan. Kompleksitas total dalam kasus terbaik:

$$O(N! \times N)$$

Untuk papan 9×9 , dengan early pruning yang efektif, jumlah iterasi dapat berkurang drastis menjadi ribuan hingga ratusan ribu iterasi (dibandingkan ratusan juta pada strategi 2).

III. Implementasi Program

A. Struktur Folder Repository

```
├── docs
│   └── LaporanStima_Tucill1.pdf
├── test
│   ├── test_1.txt
│   ├── test_2.txt
│   ├── test_3.txt
│   ├── test_4.txt
│   ├── test_5.txt
│   ├── solus_test_1.txt
│   ├── solus_test_2.txt
│   ├── solus_test_3.txt
│   ├── solus_test_4.txt
│   └── solusi_test_5.txt
├── src
│   └── real.go
├── bin
│   └── real.exe
└── README.md
```

Repository program ini disusun dengan tujuan agar seluruh komponen program terorganisir dengan rapi dan mudah dipahami. Folder src digunakan untuk menyimpan File real.go yang berisi seluruh source code program dalam satu file tunggal, mencakup fungsi utama, fungsi solver untuk ketiga strategi, dan fungsi-fungsi helper untuk validasi dan I/O.

Folder test digunakan untuk menyimpan file input yang akan diuji oleh program. File-file dalam folder ini berisi konfigurasi papan awal yang akan diproses oleh algoritma. Apabila solusi ditemukan dan pengguna memilih untuk menyimpannya, maka file solusi akan disimpan di folder yang sama dengan nama solusi_<nama_input>.txt.

Folder bin digunakan untuk menyimpan executable file .exe untuk memudahkan proses run program.

Folder docs menyimpan dokumentasi program yaitu laporan ini. Struktur ini dibuat agar program memiliki pemisahan yang jelas antara komponen sehingga lebih mudah dalam proses pengembangan, pengujian, dan pemeliharaan.

Terakhir, file README.md menyimpan dokumentasi singkat dalam bentuk ekstensi markdown yang berisi penjelasan singkat mengenai program, requirement program, cara menjalankan program, dan identitas pembuat.

B. Source Code

Program ini ditulis dalam bahasa Go (Golang) dan menggunakan paradigma procedural dengan pendekatan recursive backtracking. Seluruh fungsi disusun dalam satu file untuk kemudahan kompilasi dan deployment.

```
package main

import (
    "bufio"
    "fmt"
    "math"
    "os"
    "path/filepath"
    "strings"
    "time"
)

var pathFile string
var buffer []string
var posisiQueens [][]int
var row int
var col int
var realBoard [][]string
var solFound int = 0
var iterationCount int = 0

var pilihanStrategi int
var freqVisualisasi int

func main() {
    fmt.Println("N-QUEENS SOLVER")
}
```

```

fmt.Println("Masukkan Path Input File: ")
fmt.Scanln(&pathFile)

board := readInput(pathFile)
if board == nil {
    fmt.Println("\nProgram dihentikan karena input tidak valid")
    return
}

if row != col {
    fmt.Println("Input tidak valid")
    return
}

// pilihan optimasi
fmt.Println("Pilih Strategi Penyelesaian")
fmt.Println("1. Pure Brute Force")
fmt.Println("2. One Queen Per Column (optimasi kolom)")
fmt.Println("3. Early Pruning + optimasi kolom")
fmt.Println("Pilihan (1-3): ")
fmt.Scanln(&pilihanStrategi)

if pilihanStrategi < 1 || pilihanStrategi > 3 {
    fmt.Println("Pilihan tidak valid")
    return
}

// pilihan visualisasi
fmt.Println("Visualisasi setiap berapa iterasi? (contoh: 1000000): ")
fmt.Scanln(&freqVisualisasi)

if freqVisualisasi <= 0 {
    freqVisualisasi = 1000000
}

// init working board
tempBoard := make([][]string, row)
for i := range tempBoard {
    tempBoard[i] = make([]string, col)
}
initQueens(tempBoard)

namaStrategi := []string{"", "Pure Brute Force", "One Queen Per
Column", "Early Pruning"}
fmt.Printf("Strategi yang dipakai: %s\n",
namaStrategi[pilihanStrategi])

startTime := time.Now()
var found bool

switch pilihanStrategi {

```

```

    case 1:
        found = solveBruteForce(tempBoard, 0, 0)
    case 2:
        found = solveQueenKolom(tempBoard, 0)
    case 3:
        found = solvePruning(tempBoard, 0)
    }

    execTime := time.Since(startTime)

    if !found {
        fmt.Println("Tidak ada solusi ditemukan")
    } else {
        fmt.Println("Solusi ditemukan!")

        var save string
        fmt.Print("Apakah ingin menyimpan solusi? (y/n): ")
        fmt.Scanln(&save)
        if save == "y" || save == "Y" {
            saveToFile(tempBoard, execTime, iterationCount)
        }
    }

    fmt.Printf("Waktu Eksekusi: %.3f ms\n",
float64(execTime.Microseconds())/1000.0)
    fmt.Printf("Total Iterasi: %d\n", iterationCount)
}

func readInput(path string) [][]string {
    file, err := os.Open(path)
    if err != nil {
        fmt.Println("Error membuka file:", err)
        return nil
    }
    defer file.Close()

    scanner := bufio.NewScanner(file)
    var tempBuffer []string

    // trim whitespace
    for scanner.Scan() {
        line := scanner.Text()
        line = strings.TrimSpace(line)
        line = strings.ReplaceAll(line, " ", "")
        line = strings.ReplaceAll(line, "\t", "")

        if line != "" {
            tempBuffer = append(tempBuffer, line)
        }
    }

    if err := scanner.Err(); err != nil {
        fmt.Println("Error membaca file:", err)
        return nil
    }
}

```

```

    }

    // cek valid or invalid board
    if !isEmpty(tempBuffer) {
        return nil
    }

    tempRow := len(tempBuffer)
    tempCol := len(tempBuffer[0])

    if !isSquare(tempRow, tempCol) {
        return nil
    }

    if !isConsistentRows(tempBuffer, tempCol) {
        return nil
    }

    if !isAlphabetic(tempBuffer) {
        return nil
    }

    uniqueColors := countUniqueColors(tempBuffer)
    if !hasValidColorCount(uniqueColors, tempRow, tempCol) {
        return nil
    }

    // make board
    buffer = tempBuffer
    row = tempRow
    col = tempCol
    realBoard = make([][]string, row)

    for i := range realBoard {
        realBoard[i] = make([]string, col)
    }

    for i := range buffer {
        for j := range buffer[i] {
            realBoard[i][j] = string(buffer[i][j])
        }
    }
    return realBoard
}

// validity cekkk
func isEmpty(buffer []string) bool {
    if len(buffer) == 0 {
        fmt.Println("Error: File kosong atau tidak berisi data valid")
        return false
    }
    return true
}

```

```

func isSquare(rows, cols int) bool {
    if rows != cols {
        fmt.Printf("Error: Papan harus berbentuk persegi (N x N)\n")
        fmt.Printf("Papan saat ini: %d baris x %d kolom\n", rows, cols)
        return false
    }
    return true
}

func isConsistentRows(buffer []string, expectedLength int) bool {
    for i, line := range buffer {
        if len(line) != expectedLength {
            fmt.Printf("Error: Baris %d memiliki panjang %d, seharusnya %d\n", i+1, len(line), expectedLength)
            return false
        }
    }
    return true
}

func isAlphabetic(buffer []string) bool {
    for i, line := range buffer {
        for j, char := range line {
            if !((char >= 'A' && char <= 'Z') || (char >= 'a' && char <= 'z')) {
                fmt.Printf("Error: Karakter tidak valid '%c' di baris %d kolom %d\n", char, i+1, j+1)
                fmt.Println("Hanya karakter alphabet (A-Z, a-z) yang diperbolehkan")
                return false
            }
        }
    }
    return true
}

func countUniqueColors(buffer []string) int {
    colorSet := make(map[rune]bool)
    for _, line := range buffer {
        for _, char := range line {
            upperChar := char
            if char >= 'a' && char <= 'z' {
                upperChar = char - 32
            }
            colorSet[upperChar] = true
        }
    }
    return len(colorSet)
}

func hasValidColorCount(uniqueColors, rows, cols int) bool {
    if uniqueColors != rows {

```

```

        fmt.Printf("Error: Jumlah warna unik (%d) tidak sesuai dengan
ukuran papan (%d x %d)\n", uniqueColors, rows, cols)
        fmt.Printf("Seharusnya ada tepat %d warna yang berbeda\n", rows)
        return false
    }
    return true
}

func printBoard(board [][]string) {
    for i := range board {
        for j := range board[i] {
            fmt.Print(board[i][j])
            fmt.Print(" ")
        }
        fmt.Println()
    }
}

func initQueens(tempBoard [][]string) {
    posisiQueens = make([][]int, row)
    for i := range posisiQueens {
        posisiQueens[i] = []int{0, 0}
    }

    for i := range tempBoard {
        for j := range tempBoard[i] {
            tempBoard[i][j] = realBoard[i][j]
        }
    }
}

// pilihan 1: pure brute force
func solveBruteForce(tempBoard [][]string, idxQueen int, mulaiDari int)
bool {
    totalCell := row * col
    // base
    if idxQueen == row {
        iterationCount++

        visualize(iterationCount, tempBoard)

        if isValid(row) {
            solFound++
            fmt.Printf("\nSOLUSI DITEMUKAN\n")
            printBoard(tempBoard)
            return true
        }
        return false
    }

    // iterate
    for idx := 0; idx < totalCell; idx++ {
        r := idx / col

```

```

        c := idx % col

        if idxQueen > 0 || idx > 0 {
            barisLama := posisiQueens[idxQueen][0]
            kolomLama := posisiQueens[idxQueen][1]
            if tempBoard[barisLama][kolomLama] == "#" {
                tempBoard[barisLama][kolomLama] =
realBoard[barisLama][kolomLama]
            }
        }

        // place new queen
        tempBoard[r][c] = "#"
        posisiQueens[idxQueen] = []int{r, c}

        // rekursi
        if solveBruteForce(tempBoard, idxQueen+1, 0) {
            return true
        }
    }

    // backtrack
    barisLama := posisiQueens[idxQueen][0]
    kolomLama := posisiQueens[idxQueen][1]
    if tempBoard[barisLama][kolomLama] == "#" {
        tempBoard[barisLama][kolomLama] =
realBoard[barisLama][kolomLama]
    }
    posisiQueens[idxQueen] = []int{0, 0}
    tempBoard[0][0] = "#"

    return false
}

// pilihan 2: one queen per column
func solveQueenKolom(tempBoard [][]string, idxQueen int) bool {
    // base
    if idxQueen == row {
        iterationCount++

        visualize(iterationCount, tempBoard)

        if isValid(row) {
            solFound++
            fmt.Printf("\nSOLUSI DITEMUKAN\n")
            printBoard(tempBoard)
            return true
        }
        return false
    }

    // iterate
    idxKolom := idxQueen

```

```

        for r := 0; r < row; r++ {
            if r > 0 {
                barisLama := posisiQueens[idxQueen][0]
                kolomLama := posisiQueens[idxQueen][1]
                if tempBoard[barisLama][kolomLama] == "#" {
                    tempBoard[barisLama][kolomLama] =
realBoard[barisLama][kolomLama]
                }
            }

            // place queen
            tempBoard[r][idxKolom] = "#"
            posisiQueens[idxQueen] = []int{r, idxKolom}

            //recursif
            if solveQueenKolom(tempBoard, idxQueen+1) {
                return true
            }
        }

        //backtrack
        barisLama := posisiQueens[idxQueen][0]
        kolomLama := posisiQueens[idxQueen][1]
        if tempBoard[barisLama][kolomLama] == "#" {
            tempBoard[barisLama][kolomLama] =
realBoard[barisLama][kolomLama]
        }
        posisiQueens[idxQueen] = []int{0, idxKolom}
        tempBoard[0][idxKolom] = "#"

        return false
    }

    // pilihan 3: one queen per column + early pruning
    func solvePruning(tempBoard [][]string, idxQueen int) bool {
        // base
        if idxQueen == row {
            iterationCount++
            visualize(iterationCount, tempBoard)

            // double check validation
            if isValid(row) {
                solFound++
                fmt.Printf("\nSOLUSI DITEMUKAN\n")
                printBoard(tempBoard)
                return true
            }
            return false
        }

        idxKolom := idxQueen

        // iterate

```

```

for r := 0; r < row; r++ {
    iterationCount++

    tempBoard[r][idxKolom] = "#"
    posisiQueens[idxQueen] = []int{r, idxKolom}

    // early pruning
    if isSquareSafe(r, idxKolom, idxQueen) {
        //recursif
        if solvePruning(tempBoard, idxQueen+1) {
            return true
        }
    }

    tempBoard[r][idxKolom] = realBoard[r][idxKolom]
}

return false
}

// hhhelper func
func isSquareSafe(r, c, idxQueen int) bool {
    // cek queen sebelumnya
    for i := 0; i < idxQueen; i++ {
        rowQ := posisiQueens[i][0]
        colQ := posisiQueens[i][1]

        // cek vertikal
        if rowQ == r {
            return false
        }

        // cek horizontal
        if colQ == c {
            return false
        }

        // cek tetangga
        if isTetangga(rowQ, r, colQ, c) {
            return false
        }

        // cek warna sama
        if isWarnaSama(rowQ, colQ, r, c) {
            return false
        }
    }

    return true
}

func visualize(iterSekarang int, board [][]string) {
    if iterSekarang%freqVisualisasi == 0 {

```

```

        fmt.Printf("\nIterasi: %d\n", iterSekarang)
        printBoard(board)
    }
}

func isValid(jumlahQueen int) bool {
    for i := 0; i < jumlahQueen; i++ {
        for j := i + 1; j < jumlahQueen; j++ {
            r1 := posisiQueens[i][0]
            c1 := posisiQueens[i][1]
            r2 := posisiQueens[j][0]
            c2 := posisiQueens[j][1]

            if r1 == r2 || c1 == c2 || isTetangga(r1, r2, c1, c2) ||
isWarnaSama(r1, c1, r2, c2) {
                return false
            }
        }
    }
    return true
}

func isTetangga(row1, row2, col1, col2 int) bool {
    return math.Abs(float64(row1-row2)) <= 1 &&
math.Abs(float64(col1-col2)) <= 1
}

func isWarnaSama(row1, col1, row2, col2 int) bool {
    return realBoard[row1][col1] == realBoard[row2][col2]
}

func saveToFile(board [][]string, execTime time.Duration, iterations
int) {
    // gabungin nama inputfile ke output file path
    inputBaseName := filepath.Base(pathFile)
    inputNameOnly := strings.TrimSuffix(inputBaseName,
filepath.Ext(inputBaseName))
    outputFileName := "solusi_" + inputNameOnly + ".txt"
    outputPath := "test"
    outputFullPath := filepath.Join(outputPath, outputFileName)
    file, err := os.Create(outputFullPath)
    if err != nil {
        fmt.Println("Error membuat file:", err)
        return
    }
    defer file.Close()

    writer := bufio.NewWriter(file)
    namaStrategi := []string{"", "Pure Brute Force", "One Queen Per
Column", "Early Pruning"}
    writer.WriteString(fmt.Sprintf("Menggunakan optimasi: %s\n",
namaStrategi[pilihanStrategi]))
    for i := range board {

```

```

        for j := range board[i] {
            writer.WriteString(board[i][j])
            writer.WriteString(" ")
        }
        writer.WriteString("\n")
    }

    writer.WriteString(fmt.Sprintf("\nWaktu Eksekusi: %.3f ms\n",
float64(execTime.Microseconds())/1000.0))
    writer.WriteString(fmt.Sprintf("Total Iterasi: %d\n", iterations))

    writer.Flush()
    fmt.Println("Solusi berhasil disave ke: ", outputFullPath)
}

```

1. main()

Fungsi ini merupakan entry point program yang mengatur alur utama eksekusi. Fungsi ini bertanggung jawab untuk:

- a) Input handling: Meminta user memasukkan path file input, pilihan strategi (1-3), dan frekuensi visualisasi
- b) Validation: Memanggil readInput() dan memeriksa return value untuk memastikan validasi berhasil
- c) Error handling: Menghentikan program jika readInput() mengembalikan NIL
- d) Initialization: Membuat papan kerja dan menginisialisasi posisi queen
- e) Strategy selection: Memanggil fungsi solver yang sesuai berdasarkan pilihan user
- f) Timing: Mengukur waktu eksekusi menggunakan time.Now() dan time.Since()
- g) Output: Menampilkan hasil, statistik, dan menawarkan opsi penyimpanan

Fungsi ini menggunakan struktur switch-case untuk memilih strategi yang sesuai:

```

switch pilihanStrategi {
case 1:
    found = solveBruteForce(tempBoard, 0, 0)
case 2:
    found = solveQueenKolom(tempBoard, 0)
case 3:
    found = solvePruning(tempBoard, 0)
}

```

setelah solving selesai, program menampilkan statistik waktu eksekusi dan jumlah iterasi serta menawarkan opsi untuk menyimpan ke file jika solusi ditemukan.

2. readInput(path string) [][]string

Fungsi ini bertanggung jawab membaca file input, melakukan parse board, dan melakukan validasi input. Proses dilakukan dengan membuka file menggunakan `os.Open()`, membaca setiap baris menggunakan `bufio.Scanner`, menghapus whitespaces, skip baris kosong, dan menyimpan ke buffer sementara. Setelah itu akan memanggil fungsi fungsi validasi, `isEmpty()`, `isSquare()`, `isConsistentRows()`, `isAlphabetic()`, dan `hasValidColorCount()`. Proses dimulai dari membuat array 2D untuk `realBoard`, lalu menyalin karakter dari buffer ke papan dan menampilkan konfirmasi validasi berhasil. `RealBoard` disimpan dalam variabel global `realBoard` yang akan digunakan untuk validasi warna dan untuk restore saat backtracking. fungsi ini akan return NIL jika input invalid, dan array 2D jika valid.

3. `isEmpty(buffer []string) bool`
`isEmpty()` bertugas untuk memeriksa apakah isi file kosong, dengan cara memeriksa apakah buffer input kosong setelah proses pembacaan file. menggunakan parameter buffer yaitu array string hasil pembacaan file. Dan akan mereturn true jika buffer memiliki data dan false jika buffer kosong dan menampilkan error message.
4. `isSquare(rows,cols int) bool`
`isSquare` bertugas untuk memeriksa apakah papan berbentuk persegi (NxN). Dengan menggunakan parameter `rows` dan `cols` yaitu jumlah baris dan jumlah kolom, dan mereturn true jika `rows=cols` dan false jika `rows!=cols` dan menampilkan error message serta dimensi aktual.
5. `isConsistentRows(buffer []string, expectedLength int) bool`
`isConsistentRows()` memeriksa apakah semua baris memiliki panjang yang konsisten, dengan menggunakan parameter buffer yaitu array string berisi data board, dan `expectedLength` yaitu panjang yang diharapkan(`rows`). Dan akan me-return true jika semua baris memiliki panjang sama, dan false jika ada baris dengan panjang berbeda, serta menampilkan error message yang berisi nomor baris dan panjang aktual.
6. `isAlphabetic(buffer []string) bool`
`isAlphabetic()` memeriksa apakah semua karakter dalam board adalah alphabet (A-Z, a-z) dengan parameter buffer yaitu array string berisi data board. Proses dimulai dari iterasi melalui setiap karakter dalam setiap baris, lalu memeriksa apakah karakter berada dalam range A - Z atau a - z. Jika ditemukan karakter non-alphabet, akan menampilkan error dengan posisi aktual. Fungsi ini akan me-return true jika semua karakter adalah alphabet dan false jika ada karakter non-alphabet.
7. `countUniqueColors(buffer []string) int`
`countUniqueColors()` adalah fungsi helper untuk menghitung jumlah warna unik dalam board, dengan parameter buffer yaitu array string berisi data board. Proses dimulai dengan pembuatan map untuk menyimpan karakter unik, lalu iterasi melalui semua karakter, konversi ke uppercase untuk case-insensitive comparison, memasukkan ke map, dan return ukuran map. fungsi ini akan me-return integer jumlah warna unik.

8. `hasValidColorCount(uniqueColors, rows, cols, int) bool`
`hasValidColorCount()` memeriksa apakah jumlah warna unik sesuai dengan ukuran board, dengan parameter `uniqueColors` yaitu jumlah warna unik yang dihitung, dan `rows` serta `cols` yaitu jumlah baris dan kolom. Fungsi ini akan me-return true jika `uniqueColors == rows`, dan false jika tidak sesuai.
9. `printBoard(board [][]string)`
Fungsi ini akan menampilkan board ke CLI. Fungsi ini melakukan iterasi (nested loop) dua dimensi melalui array board untuk mencetak setiap elemen dengan spasi dan separator antar sel.
10. `initQueens(tempBoard [][]string)`
`iniQueens()` adalah fungsi inisialisasi yang mempersiapkan struktur data sebelum solving. Proses berisi pembuatan array `posisiQueens` untuk menyimpan koordinat setiap queen, inisialisasi queen ke posisi 0,0 dan menyalin `realBoard` ke `tempBoard`.
11. `solveBruteForce(tempBoard [][]string, idxQueen, mulaiDari int) bool`
`solveBruteForce()` adalah fungsi rekursif implementasi strategi 1 (pure Brute force) dengan parameter `tempBoard` yaitu working board saat ini, `idxQueen` yaitu indeks queen yang sedang ditempatkan.
Algoritma:
 - a. Base case: jika `idxQueen == row`, semua queen sudah ditempatkan.
 - i. increment `iterationCount`
 - ii. visualisasi jika diperlukan
 - iii. validasi dengan `isValid()`
 - iv. return true jika valid
 - b. Iterasi melalui semua sel papan
 - i. konversi indeks linear ke koordinat (row,col)
 - ii. hapus queen dari posisi lama
 - iii. tempatkan queen di posisi baru
 - iv. update `posisiQueens`
 - v. rekursi untuk queen berikutnya
 - vi. jika berhasil, return true
 - c. Reset queen ke 0,0 jika tidak ada solusi hingga akhir loop
return true jika solusi ditemukan dan false jika tidak ada solusi pada konfigurasi/iteration saat ini.
12. `solveQueenKolom(tempboard [][]string, idxQueen int) bool`
Fungsi rekursif ini adalah implementasi strategi one queen per column dengan parameter `tempBoard` yaitu working board saat ini dan `idxQueen` yaitu indeks queen.
Algoritma:
 - a. Base case: sama seperti `solveBruteForce`
 - b. set `idxKolom = idxQueen`
 - c. iterasi hanya melalui baris (0 sampai N-1)
 - i. hapus queen dari posisi lama (jika bukan iterasi pertama)

- ii. tempatkan queen di (r, idxkolom)
- iii. update posisiQueens
- iv. rekursi untuk queen berikutnya
- v. jika berhasil, return true

d. reset queen ke (0,idxKolom)

Perbedaan dengan strategi 1 adalah, strategi 2 hanya iterasi baris (N iterasi) bukan semua sel (N^2 iterasi) dan kolom fixed berdasarkan indeks queen

13. solvePruning(tempBoard [][]string, idxQueen int) bool

Fungsi rekursif ini adalah implementasi strategi Early Pruning dengan parameter yang sama seperti solveQueenKolom.

Algoritma:

- a. Base case: sama seperti strategi lainnya.
- b. set idxKolom = idxQueen
- c. untuk setiap baris:
 - i. increment counter iterasi
 - ii. tempatkan queen untuk visualisasi di placement sementara
 - iii. memnaggil isSquaresafe untuk cek validasi posisi queen saat ini
 - iv. jika aman, melakukan rekursi untuk queen berikutnya
 - v. backtrack: hapus semua queen untuk semua posisi bukan hanya yang valid
- d. return false jika tidak ada pososisi yang menghasilkan solusi

Key Difference: validasi dilakukan sebelum rekursi, memotong cabang / kemungkinan yang pasti invalid

14. isSquareSafe(row,col,idxQueen int) bool

Fungsi helper untuk early pruning yang memeriksa apakah posisi (row,col) aman untuk queen ke-idxQueen terhadap semua queen yang sudah ditempatkan sebelumnya (queen 0 hingga idxQueen-1)

Validasi yang dilakukan:

- a. Apakah ada queen lain di baris row
- b. Apakah ada queen lain di kolom col
- c. Apakah ada queen di posisi adjacent (isTetangga())
- d. Apakah ada queen di posisi yang berwarna sama(isWarnaSama())

Fungsi ini akan me-return true jika semua validasi lolos, dan false jika ada konflik

15. visualize(iterSekarang, board [][]string)

Fungsi visualize adalah fungsi untuk menampilkan visualisasi board pada iterasi tertentu. Fungsi ini mengecek apakah iterSekarang adalah kelipatan dari freqVisualisasi. Jika iya, tampilkan nomor iterasi dan board saat ini untuk Live Update.

16. isValid(jumlahQueen) bool

isValid adalah fungsi validasi lengkap yang dipanggil setelah semua queen ditempatkan.

Algoritma:

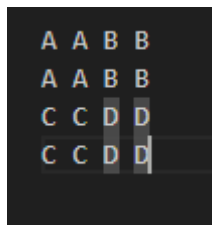
- a. iterasi melalui semua pasangan queen (i,j)
 - b. untuk setiap pasangan, ambil koordinat dan cek apakah di baris/kolom yang sama, apakah bertetangga, dan apakah memiliki warna yang sama.
 - c. jika ada pelanggaran, return false, jika semua pasangan valid, return true.
17. isTetangga(row1,row2,col1,col2 int)bool
isTetangga mengecek apakah dua posisi bersebalahan secara horizontal, vertikal dan diagonal. Implementasinya menggunakan Manhattan distance. Jika selisih baris dan kolom ≤ 1 , maka kedua posisi adjacent.
18. isWarnaSama(row1,col1,row2,col2) bool
isWarnaSama mengecek apakah dua posisi memiliki karakter/warna yang sama pada realBoard.
19. saveToFile(board [][]string,execTime,iterations int)
Fungsi ini menyimpan solusi ke File.
 - a. Generate output filename
 - i. ambil basename dari input file
 - ii. hapus .txt
 - iii. buat nama solusi_<input_name>.txt
 - iv. simpan ke folder test/
 - b. write content
 - i. menulis strategi yang digunakan
 - ii. menulis konfigurasi board
 - iii. menulis waktu eksekusi
 - iv. menulis total iterasi
 - c. Finalisasi
 - i. Flush buffer
 - ii. menampilkan pesan sukses di cli dengan path-nya

IV. Hasil Pengujian

A. Test Case 1

Test case merupakan test case valid 4x4

soal:



solusi (pure brute force, optimasi kolom, early pruning):

```

N-QUEENS SOLVER
Masukkan Path Input File:
test/test_4x4.txt
Pilih Strategi Penyelesaian
1. Pure Brute Force
2. One Queen Per Column (optimasi kolom)
3. Early Pruning + optimasi kolom
Pilihan (1-3):
1
Visualisasi setiap berapa iterasi? (contoh: 1000000):
1000
Strategi yang dipakai: Pure Brute Force

Iterasi: 1000
A A B B
A # # #
C C D D
C C D D

Iterasi: 2000
A A B B
A A B B
C C D #
C # # D

Iterasi: 3000
A # # B
A A # B
C C D #
C C D D

SOLUSI DITEMUKAN DI ITERASI KE-3812
A # B B
A A B #
# C D D
C C # D
Solusi ditemukan!
Apakah ingin menyimpan solusi? (y/n): y
Solusi berhasil disave ke: test\solusi_test_4x4.txt
Waktu Eksekusi: 2.775 ms
Total Iterasi: 3812

```

```

N-QUEENS SOLVER
Masukkan Path Input File:
test/test_4x4.txt
Pilih Strategi Penyelesaian
1. Pure Brute Force
2. One Queen Per Column (optimasi kolom)
3. Early Pruning + optimasi kolom
Pilihan (1-3):
2
Visualisasi setiap berapa iterasi? (contoh: 1000000):
100
Strategi yang dipakai: One Queen Per Column

Iterasi: 100
A A # B
# A B B
C # D D
C C D #

SOLUSI DITEMUKAN DI ITERASI KE-115
A A # B
# A B B
C C D #
C # D D
Solusi ditemukan!
Apakah ingin menyimpan solusi? (y/n): y
Solusi berhasil disave ke: test\solusi_test_4x4.txt
Waktu Eksekusi: 2.736 ms
Total Iterasi: 115

```

```

N-QUEENS SOLVER
Masukkan Path Input File:
test/test_4x4.txt
Pilih Strategi Penyelesaian
1. Pure Brute Force
2. One Queen Per Column (optimasi kolom)
3. Early Pruning + optimasi kolom
Pilihan (1-3):
3
Visualisasi setiap berapa iterasi? (contoh: 1000000):
100
Strategi yang dipakai: Early Pruning

SOLUSI DITEMUKAN DI ITERASI KE-27
A A # B
# A B B
C C D #
C # D D
Solusi ditemukan!
Apakah ingin menyimpan solusi? (y/n): y
Solusi berhasil disave ke: test\solusi_test_4x4.txt
Waktu Eksekusi: 1.162 ms
Total Iterasi: 27

```

B. Test Case 2

test case berisi board dengan konfigurasi warna kurang dari jumlah kolom dan baris soal:

```

A A A B
A B B B
A B B B
A A A B

```

jawaban:

```

N-QUEENS SOLVER
Masukkan Path Input File:
test/test_kekurangan_warna.txt
Error: Jumlah warna unik (2) tidak sesuai dengan ukuran papan (4 x 4)
Seharusnya ada tepat 4 warna yang berbeda

Program dihentikan karena input tidak valid

```

C. Test Case 3

test case berisi board dengan konfigurasi invalid input, ada selain alphabet di inputnya.

soal:

```

AAABBCCC.
ABBBBC.CD
ABBBDCED
AAABDCCLD
BBB.DDDDD
FG]GDDHDD
FGIGDDHDD
FG'GDDHDD
FGGGDDHHH

```

jawaban:

```

N-QUEENS SOLVER
Masukkan Path Input File:
test/test_titik.txt
Error: Karakter tidak valid '.' di baris 1 kolom 9
Hanya karakter alphabet (A-Z, a-z) yang diperbolehkan

Program dihentikan karena input tidak valid

```

D. Test Case 4

test case berisi board dengan konfigurasi whitespaces.

soal:

```

AAABBCCCD

ABBBBCECD
ABBBDCED
AAABDC      CCD
BB          BBDDDDD

FGGGDDHDD
FGI  GDDHDD
FGIGDDHDD
FGGGDDHHH

```

jawaban:

```

N-QUEENS SOLVER
Masukkan Path Input File:
test/test whitespaces_bolong_bolong.txt
Pilih Strategi Penyelesaian
1. Pure Brute Force
2. One Queen Per Column (optimasi kolom)
3. Early Pruning + optimasi kolom
Pilihan (1-3):
2
Visualisasi setiap berapa iterasi? (contoh: 1000000):
10000000
Strategi yang dipakai: One Queen Per Column

Iterasi: 10000000
A A A B B C C C #
A B B # B C # C D
# # B B D C E C D
A A A B D C C C D
B B B B # D D D D
F G G G D # H D D
F G I G D D H D D
F G I G D D H D D
F G # G D D H # H

Iterasi: 20000000
A A A B # C C C D
A B B B B # E C #
A B B B D C E C D
A A A # D C # C D
# B B B D D D D D
F # G G D D H D D
F G I G D D H D D
F G # G D D H # D
F G G G D D H H H

SOLUSI DITEMUKAN DI ITERASI KE-276654213
A A A B B C C # D
A B B B # C E C D
A B B B D C # C D
A # A B D C C C D
B B B B D # D D D
F G G # D D H D D
# G I G D D H D D
F G # G D D H D D
F G G G D D H H #
Solusi ditemukan!
Apakah ingin menyimpan solusi? (y/n): y
Solusi berhasil disave ke: test\solusi_test whitespaces_bolong_bolong.txt
Waktu Eksekusi: 15725.314 ms
Total Iterasi: 276654213

```

E. Test Case 5

test case berisi board dengan konfigurasi valid menurut spesifikasi tugas kecil ini.
soal:

```

AAABBCCCD
ABBBBCECD
ABBBDCEDC
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH

```

jawaban:

```

N-QUEENS SOLVER
Masukkan Path Input File:
test/test_valid.txt
Pilih Strategi Penyelesaian
1. Pure Brute Force
2. One Queen Per Column (optimasi kolom)
3. Early Pruning + optimasi kolom
Pilihan (1-3):
2
Visualisasi setiap berapa iterasi? (contoh: 1000000):
10000000
Strategi yang dipakai: One Queen Per Column

Iterasi: 10000000
A A A B B C C C #
A B B # B C # C D
# B B D C E C D
A A A B D C C C D
B B B B # D D D D
F G G G D # H D D
F G I G D D H D D
F G I G D D H D D
F G # G D D H # H

Iterasi: 20000000
A A A B # C C C D
A B B B B # E C #
A B B B D C E C D
A A A # D C # C D
# B B B D D D D D
F # G G D D H D D
F G I G D D H D D
F G # G D D H # D
F G G G D D H H H

SOLUSI DITEMUKAN DI ITERASI KE-276654213
A A A B B C C # D
A B B B # C E C D
A B B B D C # C D
A # A B D C C C D
B B B B D # D D D
F G G # D D H D D
# G I G D D H D D
F G # G D D H D D
F G G G D D H H #
Solusi ditemukan!
Apakah ingin menyimpan solusi? (y/n): y
Solusi berhasil disave ke: test\solusi_test_valid.txt
Waktu Eksekusi: 17299.116 ms
Total Iterasi: 276654213

```

```

N-QUEENS SOLVER
Masukkan Path Input File:
test/test_valid.txt
Pilih Strategi Penyelesaian
1. Pure Brute Force
2. One Queen Per Column (optimasi kolom)
3. Early Pruning + optimasi kolom
Pilihan (1-3):
3
Visualisasi setiap berapa iterasi? (contoh: 1000000):
-1
Strategi yang dipakai: Early Pruning

SOLUSI DITEMUKAN DI ITERASI KE-21925
A A A B B C C # D
A B B B # C E C D
A B B B D C # C D
A # A B D C C C D
B B B B D # D D D
F G G # D D H D D
# G I G D D H D D
F G # G D D H D D
F G G G D D H H #
Solusi ditemukan!
Apakah ingin menyimpan solusi? (y/n): y
Solusi berhasil disave ke: test\solusi_test_valid.txt
Waktu Eksekusi: 7.180 ms
Total Iterasi: 21925

```

V. Kesimpulan

A. Inefisiensi Brute Force

Algoritma Pure Brute Force bekerja dengan mengeksplorasi ruang pencarian tanpa memanfaatkan struktur permasalahan. Dengan kompleksitas $(N^2)^N$, algoritma ini mencoba setiap kemungkinan penempatan tanpa memvalidasi parsial. Untuk Board ukuran 9x9, dibutuhkan sekitar 2×10^{17} kombinasi. Sebagai perbandingan kontras, strategi early pruning yang memotong cabang invalid lebih awal dapat menemukan solusi hanya dalam kurang lebih 1000 iterasi dan waktu 3,5ms. Perbedaan ekstrim ini menggarisbawahi kelemahan fatal brute force, yaitu ketiadaan mekanisme untuk menghentikan eksplorasi pada jalur yang sudah pasti salah.

B. Dampak Penundaan Validasi

Kelemahan kritis lainnya adalah penundaan validasi hingga seluruh N queen ditempatkan. Brute force murni baru memeriksa validity setelah konfigurasi lengkap terbentuk, yang berarti brute force menghabiskan waktu menyusun banyak konfigurasi yang sebenarnya tidak mungkin benar.

VI. Kesimpulan

Implementasi algoritma brute force pada permasalahan N-Queens menegaskan bahwa pendekatan exhaustive search tanpa optimasi memiliki keterbatasan skalabilitas yang parah. Meskipun secara teoritis benar karena pasti menemukan solusi jika ada, tapi kompleksitas waktu eksponensial membuat algoritma tersebut tidak praktis untuk N yang besar.

Ketidakefisienan utama berasal dari eksplorasi buta terhadap searching space dan validasi akhir. Algoritma menghabiskan sebagian besar komputasi untuk menelusuri cabang-cabang keputusan yang sudah pasti invalid.

VII. Pranala Repository

https://github.com/alfarabi0642/Tucil1_13524086

VIII. Pernyataan

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



Al Farabi

LAMPIRAN

No	Poin	ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Solusi yang diberikan program benar dan memenuhi 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		✓

