Laporan Tugas Kecil 1

IF2211 Strategi Algoritma

Penyelesaian Permainan Queens

Semester II Tahun 2025/2026

Disusun oleh:

Jason Edward Salim - 13524034

Laboratorium Ilmu dan Rekayasa Komputasi

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Daftar Isi

1 Penerapan Algoritma Brute Force

1.1 Pendahuluan

Algoritma *Brute force* merupakan metode penyelesaian permasalahan secara *straightforward*. Pada program penyelesaian permainan Queens ini, algoritma *Brute force* digunakan secara murni dengan *exhaustive search*.

1.2 Alur Berpikir

Queens memiliki area bujur sangkar berukuran dinamis NxN dimana kita menempatkan ratu pada tiap warna dimana setiap ratu yang ditempatkan dilarang berada pada baris dan kolom yang sama, serta dilarang berada di 8 sel di sekitar ratu yang lain.
Untuk mendapatkan solusi dari Queens, *Brute force* murni diterapkan dengan menggunakan *exhaustive search* pada tiap baris secara terus-menerus hingga ditemukan solusi yang tepat. Pada setiap baris, penempatan ratu akan dilakukan pada tiap kolom hingga baris terakhir dan dilakukan pengecekan apakah penempatan ratu sekarang sesuai dengan aturan yang ada atau tidak. Jika belum sesuai, maka penempatan berikutnya akan dilakukan hingga ditemukan solusi yang sesuai atau tidak ditemukan solusi setelah seluruh penempatan dicoba. Solusi ini akan memiliki kompleksitas waktu sekitar $O(N^N)$ karena pada tiap N baris terdapat N kolom yang dapat dicoba penempatannya.

1.3 Implementasi Penyelesaian Brute Force

Pada penyelesaian ini, algoritma akan terbagi menjadi:

1. Fungsi utama yang akan melakukan *exhaustive search* untuk penempatan ratu pada tiap baris dan kolom.
2. Fungsi validasi penempatan yang akan mengecek apakah penempatan ratu pada suatu posisi sudah sesuai dengan aturan yang ada.
3. Fungsi yang akan menyimpan dan mengembalikan solusi yang didapatkan.

1.3.1 Exhaustive Search

Untuk setiap baris, kita akan menempatkan ratu pada setiap kolom dan melakukan validasi setelah tiap baris memiliki satu ratu. Langkah ini dapat diimplementasikan sebagai berikut.

1. Iterasi tiap baris dengan kombinasi tiap kolom.

   Contoh: (0, 0) hingga (0, N-1), (1, 0) hingga (1, N-1), dan seterusnya.

```
1 func TryPosition(area *TArea) (queensLocation []TPosition) {
2     n := area.n
3     cols := make([]int, n)
4     for {
5         temp := make([]TPosition, n)
6         for row := 0; row < n; row++ {
7             temp[row] = TPosition{row, cols[row]}
8         }
```

```
9
10          if CheckPosition(*area, temp) {
11              area.queensLocation = temp
12              return temp
13          }
14
15          var i int
16          for i = n - 1; i >= 0; i- {
17              cols[i]++
18              if cols[i] < n {
19                  break
20              }
21              cols[i] = 0
22          }
23          if i < 0 {
24              break
25          }
26      }
27
28      return nil
29  }
```

2. Setelah menempatkan ratu di tiap baris, validasi apakah penempatan sudah memenuhi aturan. Jika belum, iterasi ke penempatan selanjutnya.

3. Jika solusi ditemukan, solusi akan dikembalikan dan jika tidak maka akhiri iterasi.

   Pada kode di atas, fungsi *CheckPosition* memvalidasi penempatan dan jika valid maka solusi akan tersimpan pada *area.queensLocation* dan dikembalikan. Sebaliknya, kode akan berlanjut jika *CheckPosition* mengembalikan false.

Versi Optimasi

Selain brute force penuh, terdapat *TryPositionOptimized* yang menggunakan permutasi kolom. Implementasi ini mengurangi kompleksitas waktu $O(N^N)$ menjadi $O(N!)$ dengan memastikan tidak ada dua ratu berbagi kolom sejak awal.

```
1  func TryPositionOptimized(area *TArea, onStep func([]TPosition) bool)
       []TPosition {
2      n := area.n
3      cols := make([]int, n)
4      for i := 0; i < n; i++ {
5          cols[i] = i
6      }
7
8      for {
9          temp := make([]TPosition, n)
10         for row := 0; row < n; row++ {
11             temp[row] = TPosition{row, cols[row]}
12         }
13
14         if !onStep(temp) {
15             return nil
```

```
        }

        if CheckPosition(*area, temp) && OneQueen(area, temp) {
            area.queensLocation = temp
            return temp
        }

        // Next permutation
        i := n-2
        for i >= 0 && cols[i] > cols[i+1] {
            i--
        }
        if i < 0 {
            break
        }

        j := n-1
        for cols[j] < cols[i] {
            j--
        }

        cols[i], cols[j] = cols[j], cols[i]

        for l, r := i+1, n-1; l < r; l, r = l+1, r-1 {
            cols[l], cols[r] = cols[r], cols[l]
        }
    }

    return nil
}
```

Algoritma:

1. Inisialisasi kolom sebagai permutasi [0, 1, ..., N-1].
2. Untuk tiap permutasi, buat calonnya lalu validasi.
3. Jika valid dan setiap region berisi satu ratu, kembalikan solusi.
4. Hitung permutasi berikutnya dengan algoritma tersebut.
5. Jika tidak ada permutasi lain, maka tidak ada solusi.

Dengan pendekatan ini, ruang pencarian berkurang dibandingkan dengan penerapan brute force murni.

1.3.2 Validasi Penempatan

Validasi yang harus dilakukan pada setiap masukan dan penempatan ratu antara lain:

1. Cek apakah area yang dimasukkan memiliki ukuran NxN dengan N > 0.
2. Cek apakah area NxN memiliki N representasi huruf.

3. Cek apakah tiap representasi huruf selalu terhubung dan tidak terputus.
4. Cek apakah penempatan ratu sesuai dengan aturan permainan: tidak berada pada baris dan kolom yang sama serta tiap ratu tidak terletak di 8 sel sekitar ratu lainnya.

Validasi ukuran dan karakter (*InputCells*)

```
func InputCells(cells string) (area *TArea, err error) {
    cells = strings.TrimSpace(cells)
    rows := strings.Split(cells, "\n")

    rowClean := make([]string, 0, len(rows))
    for _, row := range rows {
        fix := strings.TrimSpace(row)
        if fix != "" {
            rowClean = append(rowClean, fix)
        }
    }

    if len(rowClean) == 0 || len(rowClean[0]) == 0 {
        return nil, fmt.Errorf("Input kosong.")
    }
    N := len(rowClean[0])

    if len(rowClean) != N {
        return nil, fmt.Errorf("Ukuran harus NxN.")
    }
    for _, row := range rowClean {
        if len(row) != N {
            return nil, fmt.Errorf("Ukuran harus NxN.")
        }
        for _, char := range row {
            if !unicode.IsLetter(char) {
                return nil, fmt.Errorf("Tiap sel hanya dapat
                    direpresentasikan oleh alfabet.")
            }
        }
    }
    ...
}
```

Algoritma:

1. Trim input dan buang baris kosong.
2. Ambil N berdasarkan baris pertama.
3. Validasi jumlah baris dan kolom.
4. Validasi tiap karakter adalah alfabet A-Z.

Validasi region tersambung (*ValidRegion*)

```go
func ValidRegion(area *TArea) error {
    colorPosition := make(map[byte][]TPosition)
    for _, cell := range area.cells {
        region := byte(cell.color[0])
        colorPosition[region] = append(colorPosition[region],
            cell.TPosition)
    }

    for region, position := range colorPosition {
        if len(position) == 0 {
            continue
        }

        connect := make([]bool, len(position))
        connect[0] = true

        changed := true
        for changed {
            changed = false
            for i := 0; i < len(position); i++ {
                if connect[i] {
                    continue
                }

                for j := 0; j < len(position); j++ {
                    if !connect[j] {
                        continue
                    }
                    if checkAdjacent(position[i], position[j]) {
                        connect[i] = true
                        changed = true
                        break
                    }
                }
            }
        }

        for i := 0; i < len(position); i++ {
            if !connect[i] {
                return fmt.Errorf("Region '%c' tidak boleh terpisah.",
                    region)
            }
        }
    }
    return nil
}
```

Algoritma:

1. Kelompokkan sel berdasarkan alfabet (region).
2. Untuk tiap region, cek keterhubungan 4-arah.

3. Jika ada sel yang tidak terhubung, kembalikan error.

Validasi posisi ratu (*CheckPosition* dan *OneQueen*)

```
func CheckPosition(area TArea, queensLocation []TPosition) (valid bool) {
    if len(queensLocation) == 0 {
        return true
    }

    for i := 0; i < len(queensLocation); i++ {
        p := queensLocation[i]
        for j := i + 1; j < len(queensLocation); j++ {
            q := queensLocation[j]
            if p.row == q.row {
                return false
            }
            if p.col == q.col {
                return false
            }
            if p.row-p.col == q.row-q.col {
                return false
            }
            if p.row+p.col == q.row+q.col {
                return false
            }
        }
    }
    return true
}

func OneQueen(area *TArea, queens []TPosition) bool {
    regionCount := make(map[string]int, area.totalColor)
    for _, cell := range area.cells {
        regionCount[cell.color] = 0
    }

    for _, q := range queens {
        cell := area.cells[q.row*area.n+q.col]
        regionCount[cell.color]++
    }

    for _, c := range regionCount {
        if c != 1 {
            return false
        }
    }
    return true
}
```

Algoritma:

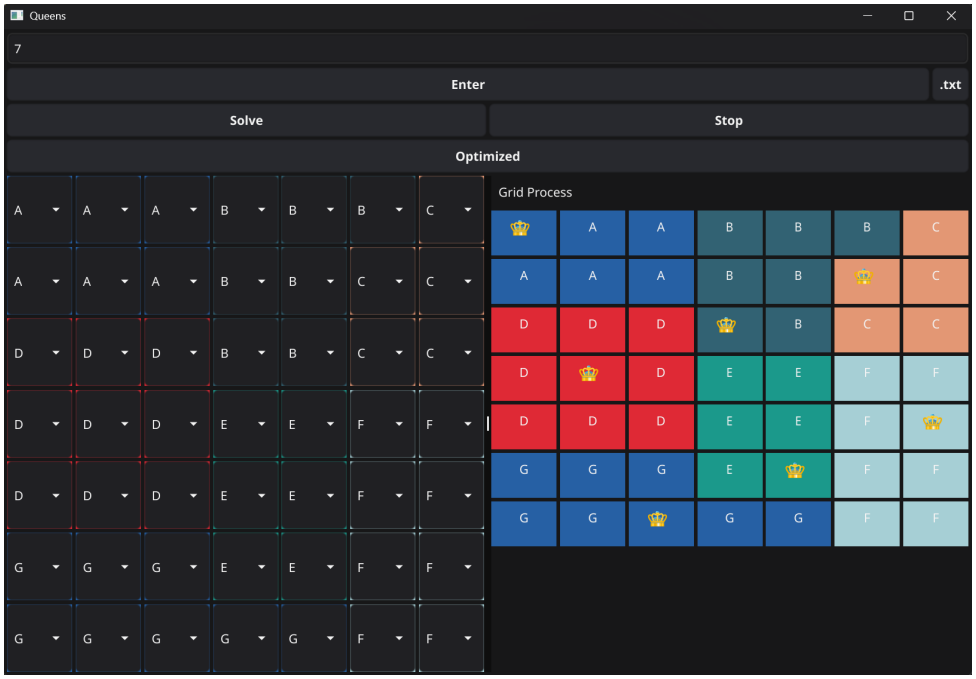
1. Pastikan tiap ratu tidak berbagi baris, kolom, atau diagonal.
2. Pastikan tiap ratu hanya ada satu pada setiap region.

Implementasi lebih lengkap dapat dilihat pada folder ***src/solution*** di *repository* GitHub yang dapat diakses pada lampiran.

2 Implementasi GUI

GUI dibuat menggunakan pustaka Fyne pada bahasa Go. Antarmuka ini bertujuan memudahkan pengguna dalam memasukkan layout, mencari solusi, serta melihat proses pencarian solusi secara visual. Seluruh komponen GUI terletak pada folder ***src/display*** dengan fungsi utama *DisplayUI* sebagai titik masuk.

2.1 Desain Antarmuka



Gambar 1: Desain Antarmuka

2.2 Komponen Utama

Komponen yang ditampilkan pada antarmuka antara lain:

1. **Input ukuran N** untuk display layout NxN
2. **Tombol .txt** untuk memuat layout dari file .txt
3. **Grid input** berupa pilihan huruf dengan warna region agar mudah dibedakan.
4. **Tombol Solve/Optimized** untuk mencari solusi brute force dan versi optimized.
5. **Tombol Stop** untuk menghentikan proses pencarian yang sedang berjalan.
6. **Panel visualisasi** yang menampilkan board dan posisi ratu selama proses pencarian.

2.3 Alur Interaksi

Pengguna dapat memilih salah satu dari dua cara input: (1) mengisi ukuran N lalu menyusun grid secara manual, atau (2) memuat layout melalui file .txt. Setelah layout valid, tombol Solve/Optimized mencari solusi di goroutine agar GUI tetap responsif. Setiap beberapa langkah, tampilan board diperbarui untuk memvisualisasikan kandidat solusi. Jika solusi ditemukan, posisi ratu ditampilkan dan hasil disimpan ke file output. Jika tidak ada solusi, GUI menampilkan pesan dan tetap menyimpan keluaran berupa layout dengan teks "Tidak ada solusi".

2.4 Implementasi Komponen GUI

Grid input dengan warna region

```
buildSelectGrid := func(n int, options []string) {
    cells := make([]*widget.Select, 0, n*n)
    rects := make([]*canvas.Rectangle, 0, n*n)
    grid := container.NewGridWithColumns(n)

    for i := 0; i < n*n; i++ {
        cell := widget.NewSelect(options, nil)
        cell.PlaceHolder = "-"

        bgRect := canvas.NewRectangle(&color.NRGBA{R: 255, G: 255, B: 255,
            A: 255})
        bgRect.SetMinSize(fyne.NewSize(50, 30))
        rects = append(rects, bgRect)

        j := i
        cell.OnChanged = func(selected string) {
            if selected != "" {
                if hexColor, ok := solution.CellColor[selected]; ok {
                    rects[j].FillColor = HexToRgb(hexColor)
                }
                rects[j].Refresh()
            }
        }

        cellContainer := container.NewStack(bgRect, cell)
        cells = append(cells, cell)
        grid.Add(cellContainer)
    }

    gridCells = cells
    gridBg = rects
    gridN = n
}
```

Algoritma:

1. Buat grid NxN dengan dropdown pilihan huruf per sel.
2. Tambahkan background persegi untuk warna region.

3. Daftarkan callback *OnChanged* untuk update warna saat huruf dipilih.
4. Stack background dan dropdown agar warna terlihat di belakang pilihan.

Memuat layout dari file .txt

```
inputTextFunc := NewFileOpen(func(closer fyne.URIReadCloser, err error) {
    if err != nil {
        dialog.NewInformation("Error", err.Error(), w).Show()
        return
    }
    defer closer.Close()

    b, readErr := io.ReadAll(closer)
    if readErr != nil {
        dialog.NewInformation("Error", readErr.Error(), w).Show()
        return
    }

    content := strings.TrimSpace(string(b))
    if content == "" {
        dialog.NewInformation("Error", "File kosong.", w).Show()
        return
    }

    if _, parseErr := updateGrid(content); parseErr != nil {
        dialog.NewInformation("Error", parseErr.Error(), w).Show()
        return
    }
}, w)
```

Algoritma:

1. Buka dialog pemilihan file .txt.
2. Baca isi file dan trim whitespace.
3. Parse layout ke grid dengan *updateGrid*.
4. Tampilkan error jika parsing gagal.

Menjalankan algoritma pencarian solusi dengan goroutine

```
runSolver := func(solver func(*solution.TArea, func([]solution.TPosition)
    bool) []solution.TPosition,
    area *solution.TArea, layout string) {

    if stopSolve != nil {
        close(stopSolve)
    }
    stopSolve = make(chan struct{})
    stopCh := stopSolve

    go func(area *solution.TArea, layout string) {
```

```
        stepCount := 0
        ans := solver(area, func(candidate []solution.TPosition) bool {
            select {
            case <-stopCh:
                return false
            default:
            }

            stepCount++
            if stepCount%50 == 0 || stepCount < 20 {
                boardArea.Objects =
                    []fyne.CanvasObject{buildVisualGrid(area, candidate)}
                boardArea.Refresh()
            }
            return true
        })

        if ans == nil {
            dialog.NewInformation("Hasil", "Tidak ada solusi untuk layout
                ini.", w).Show()
        } else {
            outputPath, _ := SaveOutput(area, ans, layout)
            dialog.NewInformation("Hasil",
                fmt.Sprintf("Solusi ditemukan setelah %d
                    langkah!\\n\\nOutput disimpan ke:\\n%s",
                stepCount, outputPath), w).Show()
        }
    }(area, layout)
}
```

Algoritma:

1. Kill goroutine lama jika ada dengan close channel.
2. Buat channel baru untuk stop signal.
3. Jalankan algoritma solusi di goroutine dengan callback update GUI tiap beberapa langkah.
4. Tampilkan hasil atau pesan "tidak ada solusi" setelah selesai.

Visualisasi layout dengan ratu

```
buildVisualGrid := func(area *solution.TArea, queens []TPosition)
    *fyne.Container {
    n := area.N()
    grid := container.NewGridWithColumns(n)

    queenMap := make(map[string]bool)
    for _, q := range queens {
        key := fmt.Sprintf("%d,%d", q.Row(), q.Col())
        queenMap[key] = true
    }
```

```
10
11      for row := 0; row < n; row++ {
12          for col := 0; col < n; col++ {
13              region := area.RegionAt(row, col)
14              key := fmt.Sprintf("%d,%d", row, col)
15              hasQueen := queenMap[key]
16
17              hexColor := area.Color(row, col)
18              bgColor := HexToRgb(hexColor)
19              bgRect := canvas.NewRectangle(bgColor)
20              bgRect.SetMinSize(fyne.NewSize(50, 50))
21
22              if hasQueen {
23                  queenText := canvas.NewText("/Emoji Mahkota/",
                        &color.NRGBA{R: 0, G: 0, B: 0, A: 255})
24                  queenText.Alignment = fyne.TextAlignCenter
25                  queenText.TextSize = 32
26                  cellContent := container.NewStack(bgRect,
                        container.NewCenter(queenText))
27                  grid.Add(cellContent)
28              } else {
29                  regionLabel := widget.NewLabel(region)
30                  regionLabel.Alignment = fyne.TextAlignCenter
31                  cellContent := container.NewStack(bgRect, regionLabel)
32                  grid.Add(cellContent)
33              }
34          }
35      }
36      return grid
37  }
```

Algoritma:

1. Buat map posisi ratu.
2. Berikan background dengan warna region untuk setiap sel.
3. Jika ada ratu tampilkan mahkota. Sebaliknya, tampilkan huruf region.
4. Kembalikan container grid untuk ditampilkan pada container visualisasi.

3 Struktur Proyek

3.1 Struktur Folder

```
1  README.md
2  bin/
3      queens.exe
4  doc/
5      13524034.pdf
6  src/
```

```
    go.mod
    main.go
    display/
        txt.go
        view.go
    solution/
        area.go
        check.go
        exha.go
test/
    input01.txt
    input02.txt
    input03.txt
    input04.txt
    input05.txt
    input06.txt
    input07.txt
    input08.txt
```

3.2 Package main

1. **main()** - titik masuk program, langsung memanggil *DisplayUI* untuk menampilkan GUI.

3.3 Package display

1. **DisplayUI()** - menggabungkan seluruh GUI: input N, load .txt, grid input, tombol Solve serta visualisasi proses.
2. **SaveOutput(area, queens, inputLayout)** - menyimpan hasil ke file *output_timestamp.txt* di folder *test*, dengan format tanda # untuk ratu.
3. **layoutToRegion(layout)** - mengambil daftar huruf region yang muncul di layout untuk opsi pilihan di grid.
4. **NewFileOpen(callback, parent)** - helper untuk dialog pemilihan file .txt di GUI.

3.4 Package solution

1. **(TArea) N()** - mengembalikan ukuran board NxN.
2. **(TArea) RegionAt(row, col)** - mengambil huruf region pada koordinat tertentu.
3. **InputCells(cells)** - parsing layout teks menjadi struktur *TArea* sekaligus validasi ukuran dan karakter.
4. **ValidRegion(area)** - mengecek ketersambungan tiap region.
5. **(TArea) Color(row, col)** - mengambil warna region untuk kebutuhan tampilan.
6. **CellColor** - tabel warna untuk tiap huruf region pada GUI.

7. **(TPosition) Row()/Col()** - getter untuk koordinat ratu.
8. **CheckPosition(area, queensLocation)** - validasi posisi ratu.
9. **PrintPosition(area, queensLocation)** - menampilkan board ke cli.
10. **OneQueen(area, queens)** - memastikan tiap region berisi tepat satu ratu.
11. **TryPosition(area, onStep)** - brute force.
12. **TryPositionOptimized(area, onStep)** - brute force dengan permutasi kolom agar proses lebih cepat.
13. **FindPosition(area)** - fungsi utama yang memanggil *TryPosition*.

4 Kasus Uji

Bagian ini berisi masukan dari file uji dan tempat untuk menempelkan tangkapan layar keluaran program.

4.1 Kasus Uji 1

Input

```
AAAA
RRRR
CCCC
DDDD
```

Output

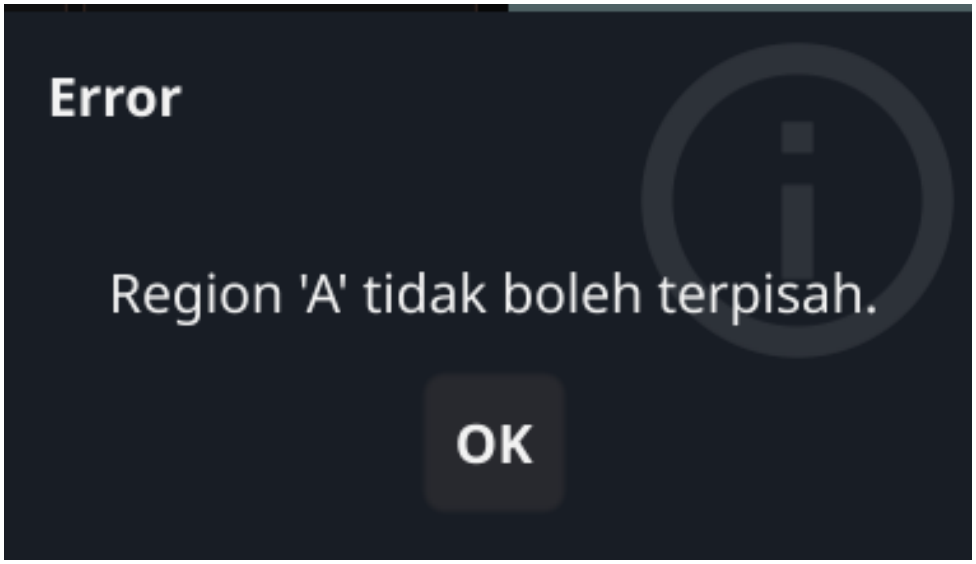
Gambar 2: Output 01

4.2 Kasus Uji 2

Input

```
AAAAA
BBBCC
CCCCC
DDDDD
DEEAA
```

Output



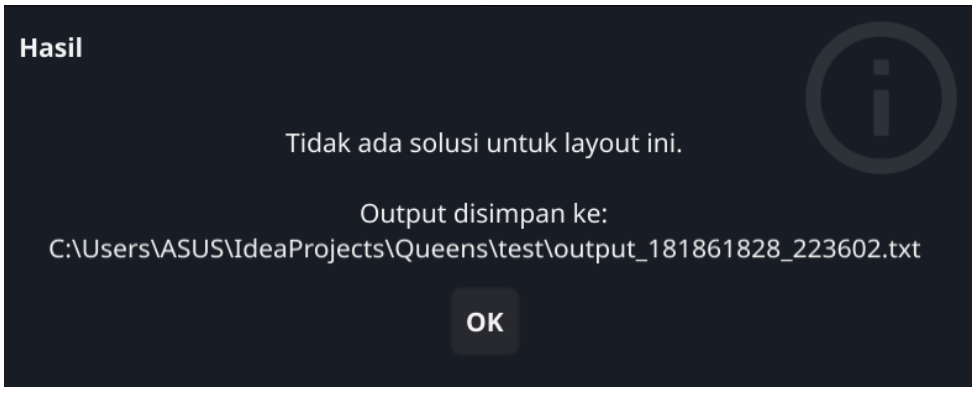
Gambar 3: Output 02

4.3 Kasus Uji 3

Input

```
AAAA
BBBB
DCDD
DDDD
```

Output



Gambar 4: Output 03

4.4 Kasus Uji 4

Input

```
AABBB
CCCCB
CCDDB
DDDBB
EEEEE
```

Output

Gambar 5: Output 04

4.5 Kasus Uji 5

Input

```
AABBB
AACCC
DDCCE
DDDEE
DDDEE
```

Output

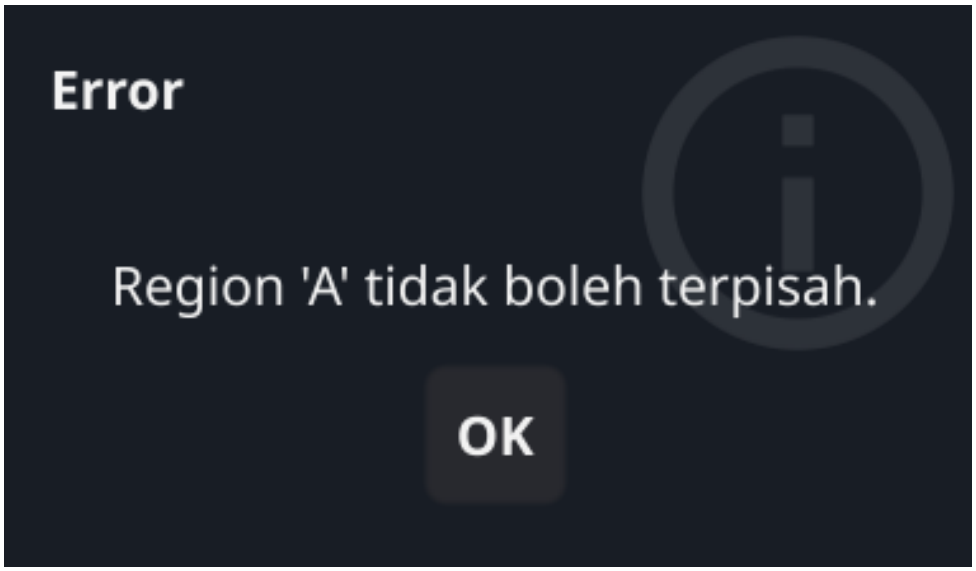
Gambar 6: Output 05

4.6 Kasus Uji 6

Input

```
AACC
DBCC
DBBB
AAAA
```

Output



Gambar 7: Output 06

4.7 Kasus Uji 7

Input

```
AAABBBC
 AAABBCC
 DDDBBCC
 DDDEEFF
 DDDEEFF
 GGGEEFF
 GGGGGFF
```

Output

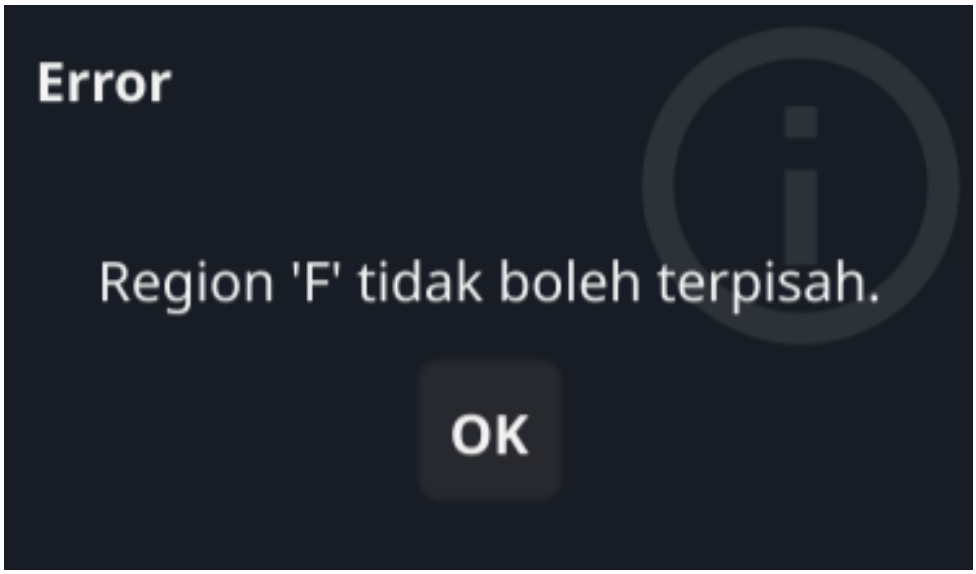
Gambar 8: Output 07

4.8 Kasus Uji 8

Input

```
AAABBC
AAABBC
DDDCCC
DDDEEE
FFDEFF
FFFEFF
```

Output



Gambar 9: Output 08

Lampiran

1. Kode program dapat diakses pada: `https://github.com/jsndwrd/Tucil1_13524034`.

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

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

Jason Edward Salim