

LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA
Kompresi Gambar dengan Metode Quadtree



Disusun oleh:

Muhammad Raihaan Perdana

13523124

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

SEMESTER GENAP 2024/2025

DAFTAR ISI

1.1. Deskripsi Program.....	2
1.2. Algoritma Program.....	3
1.2.1. Algoritma Umum Pembentukan Quadtree.....	3
1.2.2. Algoritma BuildQuadTree (Divide and Conquer).....	4
1.2.2.1. Fungsi BuildQuadTree.....	4
1.2.2.2. Fungsi DrawQuadTree.....	6
1.2.2.3. Fungsi CalculateQuadTreeStats.....	6
1.2.2.4 Fungsi FindThresholdForTargetCompression.....	7
1.2.3. Metode Perhitungan Error.....	8
1.2.3.1. Variance.....	8
1.2.3.2. Mean Absolute Deviation (MAD).....	9
1.2.3.3. Max Pixel Difference.....	9
1.2.3.4. Entropy.....	10
1.2.3.5. Structural Similarity Index (Bonus).....	11
2.1. Struktur Program.....	14
2.2. Source Code Program.....	14
2.2.1. quadtree.h.....	14
2.2.2. quadtree.cpp.....	16
2.2.3. quadtree_algorithm.cpp.....	22
2.2.4. gif_utils.cpp.....	25
2.2.5. main.cpp.....	28
3.1. Hasil Pengujian.....	34
3.1.1. Test Case 1 (Variance).....	34
3.1.2. Test Case 2 (MAD).....	35
3.1.3. Test Case 3 (Max Pixel Difference).....	35
3.1.4. Test Case 4 (Entropy).....	36
3.1.5. Test Case 5 (SSIM).....	37
3.1.6. Test Case 6 (MAD 2).....	38
3.1.7. Test Case 7 (Manual Kompresi + GIF).....	39
3.1.8. Test Case 8 (Variance 2).....	41
3.2. Analisis Pengujian.....	42
4.1 Kesimpulan.....	44
5.1. Bonus yang Diimplementasikan.....	45
5.1.1. Target Persentasi Kompresi.....	45
5.1.2. Implementasi Structural Similarity Index (SSIM).....	45
5.1.3. GIF Proses Kompresi.....	45
5.2. Pranala ke Repositori.....	45

BAB 1

DESKRIPSI PROGRAM & ALGORITMA DIVIDE AND CONQUER

1.1. Deskripsi Program

Program ini mengimplementasikan metode kompresi gambar dengan pendekatan Quadtree. Quadtree adalah struktur data hierarkis yang membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Program menerima gambar input, kemudian menerapkan algoritma divide and conquer untuk membagi gambar menjadi blok-blok berbentuk persegi yang lebih kecil. Proses pembagian dilakukan secara rekursif dan akan berhenti ketika blok memenuhi kriteria homogenitas (error berada di bawah threshold) atau mencapai ukuran minimum yang ditentukan.

Program ini memungkinkan pengguna untuk memilih berbagai metode perhitungan error seperti Variance, Mean Absolute Deviation (MAD), Max Pixel Difference, dan Entropy. Selain itu, pengguna juga dapat menetapkan ambang batas (threshold), ukuran blok minimum, dan target persentase kompresi. Setelah proses kompresi selesai, program menghasilkan gambar terkompresi dan menampilkan statistik kompresi seperti persentase kompresi, jumlah node, kedalaman pohon, serta waktu eksekusi.

Implementasi bonus pada program ini mencakup fitur penyesuaian threshold secara otomatis untuk mencapai target persentase kompresi, implementasi metode pengukuran error SSIM (Structural Similarity Index), dan visualisasi proses pembentukan Quadtree dalam format GIF.

1.2. Algoritma Program

Algoritma divide and conquer yang digunakan dalam metode kompresi Quadtree terdiri dari tiga tahap utama:

1.2.1. Algoritma Umum Pembentukan Quadtree

Algoritma kompresi gambar dengan metode Quadtree secara garis besar adalah sebagai berikut:

Masukan:

- Gambar input (I) dalam format RGB
- Threshold (T) untuk penentuan homogenitas
- Ukuran blok minimum (M)
- Metode pengukuran error (E)

Keluaran:

- Struktur Quadtree (Q) yang merepresentasikan gambar terkompresi
- Gambar terkompresi

- Statistik kompresi (waktu eksekusi, ukuran sebelum/sesudah, persentase kompresi, kedalaman pohon, jumlah node)

Algoritma:

Algoritma KompresiQuadtree merupakan algoritma utama yang mengkoordinasikan keseluruhan proses kompresi. Algoritma ini menangani penyesuaian threshold jika target persentase kompresi diaktifkan, membangun struktur Quadtree, merekonstruksi gambar terkompresi, menghitung statistik seperti waktu eksekusi dan persentase kompresi, serta menghasilkan GIF proses jika diminta. Alur algoritma dimulai dari cek parameter, pembangunan struktur data, rekonstruksi gambar, hingga penampilan hasil dan statistik.

Pseudocode:

```

procedure KompresiQuadtree(I, T, M, E, C)
    // Jika target persentase kompresi diaktifkan, cari
    threshold optimal
    if C > 0 then
        T = FindThresholdForTargetCompression(I, M, E, C)
    end if

    // Inisialisasi dan buat akar pohon Quadtree
    width = I.getWidth()
    height = I.getHeight()
    startTime = getCurrentTime()

    // Bangun pohon Quadtree
    root = BuildQuadTree(I, 0, 0, width, height, M, T, E)

    // Rekonstruksi gambar terkompresi
    outputImage = createEmptyImage(width, height)
    DrawQuadTree(outputImage, root)

    // Hitung statistik
    endTime = getCurrentTime()
    duration = endTime - startTime
    nodeCount = 0
    maxDepth = 0
    CalculateQuadTreeStats(root, nodeCount, maxDepth)

    // Hitung persentase kompresi
    originalSize = width * height * 3 // 3 bytes per piksel
    RGB
    compressedSize = nodeCount * (sizeof(node) +
    sizeof(color))
    compressionPercentage = (1 -
    compressedSize/originalSize) * 100

    // Tampilkan statistik
    print("Waktu eksekusi: ", duration, " ms")
    print("Ukuran gambar sebelum: ", originalSize, " bytes")
    print("Ukuran gambar setelah: ", compressedSize, "
    bytes")
  
```

```

    print("Persentase kompresi: ", compressionPercentage,
"%")
    print("Kedalaman pohon: ", maxDepth)
    print("Banyak simpul pada pohon: ", nodeCount)

    // Simpan gambar terkompresi
    SaveImage(outputImage, outputPath)

    // Buat GIF visualisasi proses (jika diminta)
    if gifPath not empty then
        SaveGIF(I, root, gifPath)
    end if

    return outputImage, root
end procedure

```

1.2.2. Algoritma BuildQuadTree (Divide and Conquer)

Algoritma divide and conquer untuk membangun Quadtree adalah sebagai berikut:

1.2.2.1. Fungsi BuildQuadTree

Algoritma:

Fungsi BuildQuadTree adalah inti dari algoritma divide and conquer dalam kompresi Quadtree. Fungsi ini menerima blok gambar, menghitung error berdasarkan metode yang dipilih, dan membuat keputusan apakah akan membagi blok menjadi empat kuadran atau tidak. Jika error di bawah threshold atau ukuran blok sudah minimum, fungsi akan mengembalikan node daun dengan warna rata-rata. Jika tidak, algoritma akan membagi blok menjadi empat bagian, menerapkan dirinya secara rekursif pada setiap bagian, dan menggabungkan hasilnya dalam satu node internal.

Pseudocode:

```

function BuildQuadTree(Image, x, y, width, height,
minBlockSize, threshold, method)
    // Buat node baru
    node = new QuadTreeNode(x, y, width, height)

    // Hitung warna rata-rata blok
    node.avgColor = CalculateAverageColor(Image, x, y,
width, height)

    // Hitung error berdasarkan metode yang dipilih
    error = CalculateError(Image, x, y, width, height,
method)

    // Kondisi penghentian rekursi (BASE CASE)
    if error <= threshold OR width <= minBlockSize OR height
<= minBlockSize OR
        width/2 < minBlockSize OR height/2 < minBlockSize
    then
        node.isLeaf = true
        return node
    end if

```

```

// DIVIDE: Bagi blok menjadi 4 bagian
halfWidth = width / 2
halfHeight = height / 2

// CONQUER: Rekursif untuk keempat kuadran
node.topLeft = BuildQuadTree(Image, x, y, halfWidth,
halfHeight,
                                minBlockSize, threshold,
method)

    node.topRight = BuildQuadTree(Image, x + halfWidth, y,
width - halfWidth,
                                halfHeight, minBlockSize,
threshold, method)

    node.bottomLeft = BuildQuadTree(Image, x, y +
halfHeight, halfWidth,
                                height - halfHeight,
minBlockSize, threshold, method)

    node.bottomRight = BuildQuadTree(Image, x + halfWidth, y
+ halfHeight,
                                width - halfWidth, height
- halfHeight,
                                minBlockSize, threshold,
method)

// COMBINE: Kembalikan node dengan empat anak
return node
end function

```

1.2.2.2. Fungsi DrawQuadTree

Algoritma:

Fungsi DrawQuadTree berperan dalam merekonstruksi gambar terkompresi dari struktur Quadtree. Algoritma ini melakukan traversal pada pohon dan mengisi area gambar output sesuai dengan node yang dikunjungi. Untuk node daun, seluruh area node diisi dengan warna rata-rata yang disimpan pada node tersebut. Untuk node internal, fungsi secara rekursif memproses keempat anaknya. Fungsi ini menjamin bahwa setiap piksel di gambar output diisi dengan warna yang sesuai.

```

procedure DrawQuadTree(outputImage, node)
    if node is NULL then
        return
    end if

    if node.isLeaf then
        // Isi area node dengan warna rata-rata
        for j = node.y to node.y + node.height - 1 do
            for i = node.x to node.x + node.width - 1 do
                SetPixelColor(outputImage, i, j,
node.avgColor)
            end for
        end for
    end if
end procedure

```

```

else
    // Rekursif untuk semua anak
    DrawQuadTree(outputImage, node.topLeft)
    DrawQuadTree(outputImage, node.topRight)
    DrawQuadTree(outputImage, node.bottomLeft)
    DrawQuadTree(outputImage, node.bottomRight)
end if
end procedure

```

1.2.2.3. Fungsi CalculateQuadTreeStats

Algoritma:

Fungsi CalculateQuadTreeStats menghitung statistik dari struktur Quadtree yang telah dibangun, termasuk jumlah total node dan kedalaman maksimum pohon. Algoritma ini melakukan traversal pada pohon, menghitung node saat dikunjungi, dan melacak kedalaman terdalam yang dicapai. Statistik ini penting untuk mengevaluasi efisiensi kompresi dan menganalisis karakteristik Quadtree yang terbentuk dari gambar input tertentu.

```

procedure CalculateQuadTreeStats(node, nodeCount, maxDepth,
currentDepth = 0)
    if node is NULL then
        return
    end if

    // Tambah penghitung node
    nodeCount = nodeCount + 1

    // Perbarui kedalaman maksimum jika diperlukan
    maxDepth = max(maxDepth, currentDepth)

    // Rekursif untuk node non-leaf
    if not node.isLeaf then
        CalculateQuadTreeStats(node.topLeft, nodeCount,
maxDepth, currentDepth + 1)
        CalculateQuadTreeStats(node.topRight, nodeCount,
maxDepth, currentDepth + 1)
        CalculateQuadTreeStats(node.bottomLeft, nodeCount,
maxDepth, currentDepth + 1)
        CalculateQuadTreeStats(node.bottomRight, nodeCount,
maxDepth, currentDepth + 1)
    end if
end procedure

```

1.2.2.4 Fungsi FindThresholdForTargetCompression

Algoritma:

Fungsi FindThresholdForTargetCompression mengimplementasikan pendekatan binary search untuk menemukan nilai threshold yang menghasilkan persentase kompresi mendekati target yang diinginkan. Algoritma ini bekerja dengan mencoba berbagai nilai threshold dalam suatu rentang, membangun pohon Quadtree untuk setiap nilai, menghitung persentase kompresi yang dihasilkan, dan menyesuaikan

rentang pencarian hingga nilai yang cukup akurat ditemukan atau jumlah iterasi maksimum tercapai.

```
function FindThresholdForTargetCompression(image,
minBlockSize, method, targetPercentage)
    // Inisialisasi batas atas dan bawah untuk binary search
    lowThreshold = 0.0
    highThreshold = 100.0
    currentThreshold = 0.0
    TOLERANCE = 0.5
    maxIterations = 15

    width = image.getWidth()
    height = image.getHeight()

    for i = 1 to maxIterations do
        // Coba threshold tengah dari rentang saat ini
        currentThreshold = (lowThreshold + highThreshold) /
2.0

        // Buat pohon Quadtree dengan threshold saat ini
        root = BuildQuadTree(image, 0, 0, width, height,
minBlockSize, currentThreshold,
method)

        // Hitung statistik dan persentase kompresi
        nodeCount = 0
        maxDepth = 0
        CalculateQuadTreeStats(root, nodeCount, maxDepth)
        compressionPercentage =
CalculateCompressionPercentage(image, nodeCount)

        // Bersihkan memori
        delete root

        // Cek apakah sudah cukup dekat dengan target
        if |compressionPercentage - targetPercentage| <
TOLERANCE then
            return currentThreshold
        end if

        // Sesuaikan rentang pencarian
        if compressionPercentage < targetPercentage then
            lowThreshold = currentThreshold
        else
            highThreshold = currentThreshold
        end if
    end for

    // Kembalikan threshold terbaik setelah iterasi maksimum
    return currentThreshold
end function
```


1.2.3. Metode Perhitungan Error

1.2.3.1. Variance

Algoritma:

Algoritma Variance menghitung keseragaman warna dengan mengukur deviasi dari nilai rata-rata. Pertama, algoritma menghitung nilai rata-rata (μ_R , μ_G , μ_B) untuk setiap kanal warna dengan menjumlahkan semua nilai piksel dan membaginya dengan jumlah piksel. Kemudian, untuk setiap piksel, algoritma menghitung selisih antara nilai piksel dengan rata-rata kanalnya, mengkuadratkan selisih tersebut, dan mengakumulasi hasil. Setelah semua piksel diproses, jumlah selisih kuadrat dibagi dengan jumlah piksel untuk mendapatkan variansi masing-masing kanal. Terakhir, algoritma mengembalikan rata-rata dari ketiga variansi kanal sebagai nilai error blok. Semakin tinggi nilai variansi, semakin heterogen blok tersebut.

```
function CalculateVariance(image, x, y, width, height,
avgColor)
    varR = 0, varG = 0, varB = 0
    N = width * height

    for j = y to y + height - 1 do
        for i = x to x + width - 1 do
            pixel = GetPixelColor(image, i, j)

            diffR = pixel.rgbRed - avgColor.rgbRed
            diffG = pixel.rgbGreen - avgColor.rgbGreen
            diffB = pixel.rgbBlue - avgColor.rgbBlue

            varR = varR + diffR * diffR
            varG = varG + diffG * diffG
            varB = varB + diffB * diffB
        end for
    end for

    if N > 0 then
        varR = varR / N
        varG = varG / N
        varB = varB / N
    end if

    return (varR + varG + varB) / 3.0
end function
```

1.2.3.2. Mean Absolute Deviation (MAD)

Algoritma:

Algoritma MAD memiliki alur yang mirip dengan Variance, tetapi menggunakan nilai absolut sebagai pengganti nilai kuadrat. Pertama, algoritma menghitung nilai rata-rata untuk setiap kanal RGB. Kemudian, untuk setiap piksel dalam blok, algoritma menghitung selisih absolut antara nilai piksel dan rata-rata kanalnya. Selisih absolut ini diakumulasi untuk setiap kanal dan pada akhirnya dibagi dengan jumlah piksel untuk mendapatkan MAD per kanal. Hasil akhirnya adalah rata-rata dari ketiga nilai

MAD, yang mencerminkan sebaran nilai piksel dari nilai tengahnya tanpa memberikan bobot ekstra pada penyimpangan besar seperti yang dilakukan Variance.

```
function CalculateMAD(image, x, y, width, height, avgColor)
    madR = 0, madG = 0, madB = 0
    N = width * height

    for j = y to y + height - 1 do
        for i = x to x + width - 1 do
            pixel = GetPixelColor(image, i, j)

            madR = madR + abs(pixel.rgbRed -
avgColor.rgbRed)
            madG = madG + abs(pixel.rgbGreen -
avgColor.rgbGreen)
            madB = madB + abs(pixel.rgbBlue -
avgColor.rgbBlue)
        end for
    end for

    if N > 0 then
        madR = madR / N
        madG = madG / N
        madB = madB / N
    end if

    return (madR + madG + madB) / 3.0
end function
```

1.2.3.3. Max Pixel Difference

Algoritma:

Algoritma ini bekerja dengan mencari nilai ekstrem dalam blok. Algoritma menginisialisasi variabel untuk nilai minimum dan maksimum dari setiap kanal warna (minR, maxR, minG, maxG, minB, maxB). Kemudian, algoritma memeriksa setiap piksel dalam blok, memperbarui nilai minimum dan maksimum untuk setiap kanal jika ditemukan. Setelah semua piksel diperiksa, algoritma menghitung selisih antara nilai maksimum dan minimum untuk setiap kanal (diffR, diffG, diffB). Nilai error akhir adalah rata-rata dari ketiga selisih ini. Metode ini sangat efisien karena hanya memerlukan satu kali pemindaian blok dan operasi perbandingan sederhana.

```
function CalculateMaxDifference(image, x, y, width, height)
    minR = 255, minG = 255, minB = 255
    maxR = 0, maxG = 0, maxB = 0

    for j = y to y + height - 1 do
        for i = x to x + width - 1 do
            pixel = GetPixelColor(image, i, j)

            minR = min(minR, pixel.rgbRed)
            minG = min(minG, pixel.rgbGreen)
            minB = min(minB, pixel.rgbBlue)
        end for
    end for

    diffR = maxR - minR
    diffG = maxG - minG
    diffB = maxB - minB

    return (diffR + diffG + diffB) / 3.0
end function
```

```

        maxR = max(maxR, pixel.rgbRed)
        maxG = max(maxG, pixel.rgbGreen)
        maxB = max(maxB, pixel.rgbBlue)
    end for
end for

diffR = maxR - minR
diffG = maxG - minG
diffB = maxB - minB

return (diffR + diffG + diffB) / 3.0
end function

```

1.2.3.4. Entropy

Algoritma:

Algoritma Entropy menggunakan pendekatan berbasis histogram untuk mengukur kompleksitas informasi. Pertama, algoritma menginisialisasi tiga array histogram (histR, histG, histB) dengan ukuran 256 (untuk setiap nilai warna 8-bit). Kemudian, algoritma memindai setiap piksel dalam blok dan menambahkan frekuensi kemunculan nilai piksel dalam histogram yang sesuai. Setelah itu, algoritma menghitung probabilitas setiap nilai piksel dengan membagi frekuensinya dengan jumlah total piksel. Untuk setiap probabilitas yang tidak nol, algoritma menghitung kontribusinya terhadap entropi dengan formula $-p \times \log_2(p)$. Entropi akhir untuk setiap kanal adalah jumlah dari semua kontribusi ini, dan nilai error adalah rata-rata dari entropi ketiga kanal.

```

function CalculateEntropy(image, x, y, width, height)
    histR[256] = {0}, histG[256] = {0}, histB[256] = {0}
    N = width * height

    // Hitung histogram
    for j = y to y + height - 1 do
        for i = x to x + width - 1 do
            pixel = GetPixelColor(image, i, j)

            histR[pixel.rgbRed] = histR[pixel.rgbRed] + 1
            histG[pixel.rgbGreen] = histG[pixel.rgbGreen] + 1
            histB[pixel.rgbBlue] = histB[pixel.rgbBlue] + 1
        end for
    end for

    // Hitung entropy
    entropyR = 0, entropyG = 0, entropyB = 0

    for i = 0 to 255 do
        if histR[i] > 0 then
            pR = histR[i] / N
            entropyR = entropyR - pR * log2(pR)
        end if
    end for
end function

```

```

        if histG[i] > 0 then
            pG = histG[i] / N
            entropyG = entropyG - pG * log2(pG)
        end if

        if histB[i] > 0 then
            pB = histB[i] / N
            entropyB = entropyB - pB * log2(pB)
        end if
    end for

    return (entropyR + entropyG + entropyB) / 3.0
end function

```

1.2.3.5. Structural Similarity Index (Bonus)

Algoritma:

Algoritma SSIM mengikuti beberapa tahap kompleks untuk membandingkan struktur visual. Pertama, algoritma mengekstrak nilai piksel sumber (blok asli) dan target (blok dengan warna rata-rata konstan) ke dalam array terpisah untuk setiap kanal. Kemudian, algoritma menghitung statistik untuk gambar sumber, termasuk mean (μ_X) dan variansi (σ_X^2) untuk setiap kanal, serta statistik target dengan mean (μ_Y) dari nilai rata-rata blok. Selanjutnya, algoritma menghitung kovariansi (σ_{XY}) antara sumber dan target untuk setiap kanal. Dengan statistik ini, algoritma menerapkan formula SSIM yang membandingkan luminansi (fungsi dari mean), kontras (fungsi dari variansi), dan struktur (fungsi dari kovariansi) dengan konstanta C1 dan C2 untuk stabilitas. Hasil SSIM untuk ketiga kanal digabungkan dengan pembobotan yang sesuai. Terakhir, nilai SSIM (antara 0-1) dikonversi menjadi nilai error dengan formula $(1 - \text{SSIM}) \times \text{SCALE_FACTOR}$, di mana error mendekati nol untuk gambar yang sangat mirip.

```

function CalculateSSIM(image, x, y, width, height, avgColor)
    // Konstanta untuk stabilitas
    C1 = 6.5025    // (0.01 * 255)2
    C2 = 58.5225   // (0.03 * 255)2

    // Bobot untuk setiap channel
    wR = 0.33333, wG = 0.33333, wB = 0.33334

    // Ekstrak nilai piksel sumber dan target
    sourceR = [], sourceG = [], sourceB = []
    targetR = [], targetG = [], targetB = []

    // Isi array sumber dari blok asli, target dari warna
    rata-rata
    for j = y to y + height - 1 do
        for i = x to x + width - 1 do
            pixel = GetPixelColor(image, i, j)

            sourceR.append(pixel.rgbRed)
            sourceG.append(pixel.rgbGreen)

```

```

        sourceB.append(pixel.rgbBlue)

        targetR.append(avgColor.rgbRed)
        targetG.append(avgColor.rgbGreen)
        targetB.append(avgColor.rgbBlue)
    end for
end for

// Hitung statistik untuk setiap channel warna
muX_R = 0, muX_G = 0, muX_B = 0
sigmaX2_R = 0, sigmaX2_G = 0, sigmaX2_B = 0

muY_R = avgColor.rgbRed
muY_G = avgColor.rgbGreen
muY_B = avgColor.rgbBlue
sigmaY2_R = 0, sigmaY2_G = 0, sigmaY2_B = 0

sigmaXY_R = 0, sigmaXY_G = 0, sigmaXY_B = 0

// Hitung mean untuk sumber
n = sourceR.size()
for i = 0 to n-1 do
    muX_R = muX_R + sourceR[i]
    muX_G = muX_G + sourceG[i]
    muX_B = muX_B + sourceB[i]
end for

muX_R = muX_R / n
muX_G = muX_G / n
muX_B = muX_B / n

// Hitung variansi dan kovariansi
for i = 0 to n-1 do
    diffX_R = sourceR[i] - muX_R
    diffX_G = sourceG[i] - muX_G
    diffX_B = sourceB[i] - muX_B

    sigmaX2_R = sigmaX2_R + diffX_R * diffX_R
    sigmaX2_G = sigmaX2_G + diffX_G * diffX_G
    sigmaX2_B = sigmaX2_B + diffX_B * diffX_B

    sigmaXY_R = sigmaXY_R + diffX_R * (targetR[i] -
muY_R)
    sigmaXY_G = sigmaXY_G + diffX_G * (targetG[i] -
muY_G)
    sigmaXY_B = sigmaXY_B + diffX_B * (targetB[i] -
muY_B)
end for

sigmaX2_R = sigmaX2_R / n
sigmaX2_G = sigmaX2_G / n
sigmaX2_B = sigmaX2_B / n

sigmaXY_R = sigmaXY_R / n
sigmaXY_G = sigmaXY_G / n

```

```

sigmaXY_B = sigmaXY_B / n

// Hitung SSIM untuk setiap channel
ssim_R = ((2 * muX_R * muY_R + C1) * (2 * sigmaXY_R +
C2)) /
          ((muX_R * muX_R + muY_R * muY_R + C1) *
(sigmaX2_R + sigmaY2_R + C2))

ssim_G = ((2 * muX_G * muY_G + C1) * (2 * sigmaXY_G +
C2)) /
          ((muX_G * muX_G + muY_G * muY_G + C1) *
(sigmaX2_G + sigmaY2_G + C2))

ssim_B = ((2 * muX_B * muY_B + C1) * (2 * sigmaXY_B +
C2)) /
          ((muX_B * muX_B + muY_B * muY_B + C1) *
(sigmaX2_B + sigmaY2_B + C2))

ssim = wR * ssim_R + wG * ssim_G + wB * ssim_B

// Konversi SSIM ke nilai error (SSIM 1 = perfect match
= error 0)
SCALE_FACTOR = 10000.0
error = (1.0 - ssim) * SCALE_FACTOR

return error
end function

```

BAB 2

STRUKTUR & SOURCE CODE PROGRAM

2.1. Struktur Program

Program terdiri dari beberapa folder yaitu:

1. src (source code program):
 - a. quadtree.h : Header untuk struktur data dan fungsi Quadtree
 - b. quadtree.cpp : Implementasi fungsi perhitungan error
 - c. quadtree_algorithm.cpp : Implementasi algoritma Quadtree
 - d. gif_utils.cpp : Implementasi utilitas pembuatan GIF
 - e. main.cpp : File utama yang mengatur alur program
2. lib\FreeImage (library tambahan untuk pemrosesan gambar):
 - a. FreeImage.h : Header untuk struktur data dan fungsi library FreeImage
 - b. FreeImage.lib : File library untuk FreeImage
3. bin (kumpulan executable file):
 - a. FreeImage.dll : Executable file untuk library FreeImage
 - b. quadtree_compression.exe : Executable file untuk program utama (nama bebas)
4. test : Kumpulan file input dan output gambar hasil kompresi beserta gif
5. doc : Laporan tugas
6. README.md : Informasi tentang program dan cara menjalankannya

2.2. Source Code Program

2.2.1. quadtree.h

```
#ifndef QUADTREE_H
#define QUADTREE_H

#include <vector>
#include <string>
#include "FreeImage.h"

// Struktur Node Quadtree
struct QuadTreeNode {
    int x, y; // Posisi (koordinat kiri atas)
    int width, height; // Ukuran blok
    RGBQUAD avgColor; // Warna rata-rata blok
    bool isLeaf; // Flag untuk node daun

    // Child nodes (untuk non-leaf)
    QuadTreeNode* topLeft;
    QuadTreeNode* topRight;
    QuadTreeNode* bottomLeft;
    QuadTreeNode* bottomRight;
};
```

```

// Constructor
QuadTreeNode(int _x, int _y, int _width, int _height);

// Destructor
~QuadTreeNode();
};

// Fungsi untuk perhitungan warna dan error
RGBQUAD calculateAverageColor(FIBITMAP* image, int x, int y, int width, int height);
double calculateVariance(FIBITMAP* image, int x, int y, int width, int height, RGBQUAD avgColor);
double calculateMAD(FIBITMAP* image, int x, int y, int width, int height, RGBQUAD avgColor);
double calculateMaxDifference(FIBITMAP* image, int x, int y, int width, int height);
double calculateEntropy(FIBITMAP* image, int x, int y, int width, int height);
double calculateSSIM(FIBITMAP* image, int x, int y, int width, int height, RGBQUAD avgColor);
double calculateError(FIBITMAP* image, int x, int y, int width, int height, int method);

// Fungsi untuk pembangunan dan visualisasi Quadtree
QuadTreeNode* buildQuadTree(FIBITMAP* image, int x, int y, int width, int height,
                             int minBlockSize, double threshold, int method);
void drawQuadTree(FIBITMAP* outputImage, QuadTreeNode* node);
void calculateQuadTreeStats(QuadTreeNode* node, int& nodeCount, int& maxDepth, int currentDepth = 0);
double calculateCompressionPercentage(FIBITMAP* originalImage, int nodeCount);
int getQuadTreeDepth(QuadTreeNode* node);

// Fungsi untuk mencari threshold optimal
double findThresholdForTargetCompression(FIBITMAP* image, int minBlockSize, int method, double targetPercentage);

// Fungsi untuk membuat dan menyimpan GIF menggunakan FreeImage
std::vector<FIBITMAP*> createQuadTreeFrames(FIBITMAP* image, QuadTreeNode* root);

// Fungsi untuk membuat dan menyimpan GIF menggunakan ImageMagick
bool saveGIF(FIBITMAP* originalImage, QuadTreeNode* root, const std::string& outputPath, bool useMagickExe = false);

// Metode utk pengukuran error
const char* getErrorMethodName(int method);

#endif

```


2.2.2. quadtree.cpp

```
#include "quadtree.h"
#include <cmath>
#include <functional>
#include <algorithm>

// CTOR dan DTOR
QuadTreeNode::QuadTreeNode(int _x, int _y, int _width, int _height)
    : x(_x), y(_y), width(_width), height(_height), isLeaf(false),
      topLeft(nullptr), topRight(nullptr), bottomLeft(nullptr),
      bottomRight(nullptr) {
    avgColor.rgbRed = 0;
    avgColor.rgbGreen = 0;
    avgColor.rgbBlue = 0;
    avgColor.rgbReserved = 0;
}

QuadTreeNode::~QuadTreeNode() {
    if (topLeft) delete topLeft;
    if (topRight) delete topRight;
    if (bottomLeft) delete bottomLeft;
    if (bottomRight) delete bottomRight;
}

// Fungsi untuk menghitung rata-rata warna dalam suatu blok
RGBQUAD calculateAverageColor(FIBITMAP* image, int x, int y, int width, int
height) {
    RGBQUAD avgColor = {0, 0, 0, 0};
    unsigned long long totalRed = 0, totalGreen = 0, totalBlue = 0;
    int pixelCount = 0;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            RGBQUAD pixel;
            FreeImage_GetPixelColor(image, i, j, &pixel);

            totalRed += pixel.rgbRed;
            totalGreen += pixel.rgbGreen;
            totalBlue += pixel.rgbBlue;
            pixelCount++;
        }
    }

    if (pixelCount > 0) {
        avgColor.rgbRed = static_cast<BYTE>(totalRed / pixelCount);
        avgColor.rgbGreen = static_cast<BYTE>(totalGreen / pixelCount);
        avgColor.rgbBlue = static_cast<BYTE>(totalBlue / pixelCount);
    }

    return avgColor;
}
```

```

}

// Fungsi untuk menghitung variance
double calculateVariance(FIBITMAP* image, int x, int y, int width, int
height, RGBQUAD avgColor) {
    double varR = 0, varG = 0, varB = 0;
    int N = width * height;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            RGBQUAD pixel;
            FreeImage_GetPixelColor(image, i, j, &pixel);

            double diffR = pixel.rgbRed - avgColor.rgbRed;
            double diffG = pixel.rgbGreen - avgColor.rgbGreen;
            double diffB = pixel.rgbBlue - avgColor.rgbBlue;

            varR += diffR * diffR;
            varG += diffG * diffG;
            varB += diffB * diffB;
        }
    }

    if (N > 0) {
        varR /= N;
        varG /= N;
        varB /= N;
    }

    return (varR + varG + varB) / 3.0;
}

// Fungsi untuk menghitung Mean Absolute Deviation (MAD)
double calculateMAD(FIBITMAP* image, int x, int y, int width, int height,
RGBQUAD avgColor) {
    double madR = 0, madG = 0, madB = 0;
    int N = width * height;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            RGBQUAD pixel;
            FreeImage_GetPixelColor(image, i, j, &pixel);

            madR += abs(static_cast<int>(pixel.rgbRed) - avgColor.rgbRed);
            madG += abs(static_cast<int>(pixel.rgbGreen) -
avgColor.rgbGreen);
            madB += abs(static_cast<int>(pixel.rgbBlue) - avgColor.rgbBlue);
        }
    }
}

```

```

        if (N > 0) {
            madR /= N;
            madG /= N;
            madB /= N;
        }

        return (madR + madG + madB) / 3.0;
    }

// Fungsi untuk menghitung Max Pixel Difference
double calculateMaxDifference(FIBITMAP* image, int x, int y, int width, int height) {
    BYTE minR = 255, minG = 255, minB = 255;
    BYTE maxR = 0, maxG = 0, maxB = 0;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            RGBQUAD pixel;
            FreeImage_GetPixelColor(image, i, j, &pixel);

            minR = std::min(minR, pixel.rgbRed);
            minG = std::min(minG, pixel.rgbGreen);
            minB = std::min(minB, pixel.rgbBlue);

            maxR = std::max(maxR, pixel.rgbRed);
            maxG = std::max(maxG, pixel.rgbGreen);
            maxB = std::max(maxB, pixel.rgbBlue);
        }
    }

    double diffR = maxR - minR;
    double diffG = maxG - minG;
    double diffB = maxB - minB;

    return (diffR + diffG + diffB) / 3.0;
}

// Fungsi untuk menghitung Entropy
double calculateEntropy(FIBITMAP* image, int x, int y, int width, int height)
{
    int histR[256] = {0}, histG[256] = {0}, histB[256] = {0};
    int N = width * height;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            RGBQUAD pixel;
            FreeImage_GetPixelColor(image, i, j, &pixel);

            histR[pixel.rgbRed]++;
            histG[pixel.rgbGreen]++;

```

```

        histB[pixel.rgbBlue]++;
    }
}

// Hitung entropy
double entropyR = 0, entropyG = 0, entropyB = 0;

for (int i = 0; i < 256; i++) {
    if (histR[i] > 0) {
        double pR = static_cast<double>(histR[i]) / N;
        entropyR -= pR * log2(pR);
    }

    if (histG[i] > 0) {
        double pG = static_cast<double>(histG[i]) / N;
        entropyG -= pG * log2(pG);
    }

    if (histB[i] > 0) {
        double pB = static_cast<double>(histB[i]) / N;
        entropyB -= pB * log2(pB);
    }
}

return (entropyR + entropyG + entropyB) / 3.0;
}

// Fungsi untuk menghitung SSIM
double calculateSSIM(FIBITMAP* image, int x, int y, int width, int height,
RGBQUAD avgColor) {
    // Konstanta untuk stabilitas
    const double C1 = 6.5025;    // (0.01 * 255)2
    const double C2 = 58.5225;   // (0.03 * 255)2

    // Bobot untuk setiap channel (defaultnya sama)
    const double wR = 0.33333, wG = 0.33333, wB = 0.33334;

    std::vector<double> sourceR, sourceG, sourceB;
    std::vector<double> targetR, targetG, targetB;

    // Isi gambar sumber dari blok asli
    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            RGBQUAD pixel;
            FreeImage_GetPixelColor(image, i, j, &pixel);

            sourceR.push_back(static_cast<double>(pixel.rgbRed));
            sourceG.push_back(static_cast<double>(pixel.rgbGreen));
            sourceB.push_back(static_cast<double>(pixel.rgbBlue));

```

```

        // Gambar target adalah blok dengan warna rata-rata yang sama
        targetR.push_back(static_cast<double>(avgColor.rgbRed));
        targetG.push_back(static_cast<double>(avgColor.rgbGreen));
        targetB.push_back(static_cast<double>(avgColor.rgbBlue));
    }
}

// Hitung statistik untuk setiap channel warna
// Sumber (gambar asli)
double muX_R = 0, muX_G = 0, muX_B = 0;
double sigmaX2_R = 0, sigmaX2_G = 0, sigmaX2_B = 0;

// Target (gambar dengan warna konstan rata-rata)
double muY_R = avgColor.rgbRed;
double muY_G = avgColor.rgbGreen;
double muY_B = avgColor.rgbBlue;
double sigmaY2_R = 0, sigmaY2_G = 0, sigmaY2_B = 0;

// Kovariansi
double sigmaXY_R = 0, sigmaXY_G = 0, sigmaXY_B = 0;

// Hitung mean
int n = sourceR.size();
for (int i = 0; i < n; i++) {
    muX_R += sourceR[i];
    muX_G += sourceG[i];
    muX_B += sourceB[i];
}

muX_R /= n;
muX_G /= n;
muX_B /= n;

// Hitung variansi dan kovariansi
for (int i = 0; i < n; i++) {
    double diffX_R = sourceR[i] - muX_R;
    double diffX_G = sourceG[i] - muX_G;
    double diffX_B = sourceB[i] - muX_B;

    sigmaX2_R += diffX_R * diffX_R;
    sigmaX2_G += diffX_G * diffX_G;
    sigmaX2_B += diffX_B * diffX_B;

    sigmaXY_R += diffX_R * (targetR[i] - muY_R);
    sigmaXY_G += diffX_G * (targetG[i] - muY_G);
    sigmaXY_B += diffX_B * (targetB[i] - muY_B);
}

sigmaX2_R /= n;
sigmaX2_G /= n;

```

```

    sigmaX2_B /= n;

    sigmaXY_R /= n;
    sigmaXY_G /= n;
    sigmaXY_B /= n;

    // Hitung SSIM untuk setiap channel
    double ssim_R = ((2 * muX_R * muY_R + C1) * (2 * sigmaXY_R + C2)) /
        ((muX_R * muX_R + muY_R * muY_R + C1) * (sigmaX2_R +
sigmaY2_R + C2));

    double ssim_G = ((2 * muX_G * muY_G + C1) * (2 * sigmaXY_G + C2)) /
        ((muX_G * muX_G + muY_G * muY_G + C1) * (sigmaX2_G +
sigmaY2_G + C2));

    double ssim_B = ((2 * muX_B * muY_B + C1) * (2 * sigmaXY_B + C2)) /
        ((muX_B * muX_B + muY_B * muY_B + C1) * (sigmaX2_B +
sigmaY2_B + C2));

    double ssim = wR * ssim_R + wG * ssim_G + wB * ssim_B;

    // SSIM berkisar dari 0 hingga 1, dengan 1 menunjukkan kesamaan sempurna
    // Untuk kompresi quadtree, nilai error yang tinggi = kurang mirip
    const double SCALE_FACTOR = 10000.0;

    // Jika ssim mendekati 1 (sangat mirip), maka error akan mendekati 0
    double error = (1.0 - ssim) * SCALE_FACTOR;

    return error;
}

// Fungsi untuk memilih metode error dan memanggil fungsinya
double calculateError(FIBITMAP* image, int x, int y, int width, int height,
int method) {
    RGBQUAD avgColor = calculateAverageColor(image, x, y, width, height);

    switch (method) {
        case 1:
            return calculateVariance(image, x, y, width, height, avgColor);
        case 2:
            return calculateMAD(image, x, y, width, height, avgColor);
        case 3:
            return calculateMaxDifference(image, x, y, width, height);
        case 4:
            return calculateEntropy(image, x, y, width, height);
        case 5:
            return calculateSSIM(image, x, y, width, height, avgColor);
        default:
            return calculateVariance(image, x, y, width, height, avgColor);
    }
}

```

```

}

// Fungsi untuk mendapatkan nama metode pengukuran error
const char* getErrorMethodName(int method) {
    switch (method) {
        case 1: return "Variance";
        case 2: return "MAD";
        case 3: return "Max Pixel Difference";
        case 4: return "Entropy";
        case 5: return "SSIM";
        default: return "Unknown";
    }
}
}

```

2.2.3. quadtree_algorithm.cpp

```

#include "quadtree.h"
#include <functional>
#include <cmath>
#include <iostream>
#include <algorithm>

QuadTreeNode* buildQuadTree(FIBITMAP* image, int x, int y, int width, int
height,
                                int minBlockSize, double threshold, int method) {
    QuadTreeNode* node = new QuadTreeNode(x, y, width, height);

    node->avgColor = calculateAverageColor(image, x, y, width, height);

    double error = calculateError(image, x, y, width, height, method);

    // Cek kondisi penghentian:
    // 1. Jika error di bawah threshold, blok tidak perlu dibagi lagi
    // 2. Jika ukuran blok sudah minimum, blok tidak bisa dibagi lagi
    // 3. Jika ukuran blok setelah dibagi akan lebih kecil dari minimum, blok
    tidak dibagi
    if (error <= threshold || width <= minBlockSize || height <= minBlockSize
    ||
        width/2 < minBlockSize || height/2 < minBlockSize) {
        node->isLeaf = true;
        return node;
    }

    // Divide: Bagi blok menjadi 4 bagian
    int halfWidth = width / 2;
    int halfHeight = height / 2;

    // Conquer: Rekursif untuk keempat kuadran
    node->topLeft = buildQuadTree(image, x, y, halfWidth, halfHeight,
                                minBlockSize, threshold, method);

```

```

        node->topRight = buildQuadTree(image, x + halfWidth, y, width -
halfWidth,
                                halfHeight, minBlockSize, threshold,
method);

        node->bottomLeft = buildQuadTree(image, x, y + halfHeight, halfWidth,
                                height - halfHeight, minBlockSize,
threshold, method);

        node->bottomRight = buildQuadTree(image, x + halfWidth, y + halfHeight,
                                width - halfWidth, height - halfHeight,
                                minBlockSize, threshold, method);

        return node;
    }

// Fungsi untuk menggambar Quadtree ke gambar output
void drawQuadTree(FIBITMAP* outputImage, QuadTreeNode* node) {
    if (!node) return;

    if (node->isLeaf) {
        for (int j = node->y; j < node->y + node->height; j++) {
            for (int i = node->x; i < node->x + node->width; i++) {
                FreeImage_SetPixelColor(outputImage, i, j, &node->avgColor);
            }
        }
    } else {
        // Rekursif untuk semua anak
        drawQuadTree(outputImage, node->topLeft);
        drawQuadTree(outputImage, node->topRight);
        drawQuadTree(outputImage, node->bottomLeft);
        drawQuadTree(outputImage, node->bottomRight);
    }
}

// Fungsi untuk menghitung statistik Quadtree
void calculateQuadTreeStats(QuadTreeNode* node, int& nodeCount, int&
maxDepth, int currentDepth) {
    if (!node) return;

    nodeCount++;
    maxDepth = std::max(maxDepth, currentDepth);

    if (!node->isLeaf) {
        calculateQuadTreeStats(node->topLeft, nodeCount, maxDepth,
currentDepth + 1);
        calculateQuadTreeStats(node->topRight, nodeCount, maxDepth,
currentDepth + 1);
        calculateQuadTreeStats(node->bottomLeft, nodeCount, maxDepth,
currentDepth + 1);
    }
}

```



```

        calculateQuadTreeStats(node->bottomRight, nodeCount, maxDepth,
currentDepth + 1);
    }
}

// Fungsi untuk menghitung persentase kompresi
double calculateCompressionPercentage(FIBITMAP* originalImage, int nodeCount)
{
    int width = FreeImage_GetWidth(originalImage);
    int height = FreeImage_GetHeight(originalImage);

    // Asumsi gambar RGB dimana setiap pixel membutuhkan 3 bytes
    unsigned long long originalSize = static_cast<unsigned long long>(width
* height * 3;

    unsigned long long compressedSize = static_cast<unsigned long
long>(nodeCount) *
                                (2 * sizeof(int) + 2 * sizeof(int) +
sizeof(RGBQUAD) + sizeof(bool));

    // persentase kompresi
    double compressionPercentage = (1.0 - static_cast<double>(compressedSize)
/ originalSize) * 100.0;

    return compressionPercentage;
}

// Fungsi untuk mencari threshold optimal untuk target persentase kompresi
double findThresholdForTargetCompression(FIBITMAP* image, int minBlockSize,
int method, double targetPercentage) {
    double lowThreshold = 0.0;
    double highThreshold = 100.0;
    double currentThreshold;

    const double TOLERANCE = 0.5;
    int maxIterations = 15; // Iterasi dibatasi untuk efisiensi

    int width = FreeImage_GetWidth(image);
    int height = FreeImage_GetHeight(image);

    for (int i = 0; i < maxIterations; i++) {
        currentThreshold = (lowThreshold + highThreshold) / 2.0;

        QuadTreeNode* root = buildQuadTree(image, 0, 0, width, height,
minBlockSize, currentThreshold,
method);

        int nodeCount = 0, maxDepth = 0;
        calculateQuadTreeStats(root, nodeCount, maxDepth);
    }
}

```

```

        double compressionPercentage = calculateCompressionPercentage(image,
nodeCount);

        delete root;

        if (fabs(compressionPercentage - targetPercentage) < TOLERANCE) {
            return currentThreshold;
        }

        if (compressionPercentage < targetPercentage) {
            lowThreshold = currentThreshold;
        } else {
            highThreshold = currentThreshold;
        }
    }

    return currentThreshold;
}

```

2.2.4. gif_utils.cpp

```

#include "quadtree.h"
#include <iostream>
#include <sstream>
#include <vector>
#include <string>
#include <cstdio>
#include <sys/stat.h>
#include <functional>

bool fileExists(const std::string& filename) {
    struct stat buffer;
    return (stat(filename.c_str(), &buffer) == 0);
}

// Fungsi untuk membuat frame dari quadtree untuk kedalaman tertentu
FIBITMAP* createFrameAtDepth(QuadTreeNode* root, int width, int height, int
maxDepth) {
    FIBITMAP* frameBitmap = FreeImage_Allocate(width, height, 24);
    if (!frameBitmap) {
        std::cerr << "Error: Gagal mengalokasikan memori untuk frame!" <<
std::endl;
        return nullptr;
    }

    std::function<void(QuadTreeNode*, int)> drawNodesUpToDepth =
[&frameBitmap, &drawNodesUpToDepth, maxDepth] (QuadTreeNode* node, int
depth) {
        if (!node) return;

        if (node->isLeaf || depth >= maxDepth) {

```

```

        for (int j = node->y; j < node->y + node->height; j++) {
            for (int i = node->x; i < node->x + node->width; i++) {
                FreeImage_SetPixelColor(frameBitmap, i, j,
&node->avgColor);
            }
        }
    } else {
        drawNodesUpToDepth(node->topLeft, depth + 1);
        drawNodesUpToDepth(node->topRight, depth + 1);
        drawNodesUpToDepth(node->bottomLeft, depth + 1);
        drawNodesUpToDepth(node->bottomRight, depth + 1);
    }
};

drawNodesUpToDepth(root, 0);

return frameBitmap;
}

// Fungsi untuk mendapatkan kedalaman maksimum quadtree
int getQuadTreeDepth(QuadTreeNode* node) {
    if (!node) return 0;
    if (node->isLeaf) return 0;

    int topLeftDepth = getQuadTreeDepth(node->topLeft);
    int topRightDepth = getQuadTreeDepth(node->topRight);
    int bottomLeftDepth = getQuadTreeDepth(node->bottomLeft);
    int bottomRightDepth = getQuadTreeDepth(node->bottomRight);

    int maxChildDepth = std::max(
        std::max(topLeftDepth, topRightDepth),
        std::max(bottomLeftDepth, bottomRightDepth)
    );

    return maxChildDepth + 1;
}

// Fungsi untuk menyimpan GIF
bool saveGIF(FIBITMAP* originalImage, QuadTreeNode* root, const std::string&
outputPath, bool useMagickExe) {
    int width = FreeImage_GetWidth(originalImage);
    int height = FreeImage_GetHeight(originalImage);

    int maxDepth = getQuadTreeDepth(root);
    std::cout << "Kedalaman pohon quadtree: " << maxDepth << std::endl;

    std::vector<std::string> frameFileNames;

    // buat nampilin frame original tapi di-disable biar ga ikut masuk frame
    originalnya

```

```

// std::string originalFrame = "frame_original.png";
// frameFileNames.push_back(originalFrame);
// FreeImage_Save(FIF_PNG, originalImage, originalFrame.c_str(), 0);

for (int depth = 0; depth <= maxDepth; depth++) {
    bool hasNonLeafAtThisDepth = false;

    std::function<void(QuadTreeNode*, int)> checkNode =
        [&](QuadTreeNode* node, int currentDepth) {
            if (!node || node->isLeaf) return;
            if (currentDepth == depth) hasNonLeafAtThisDepth = true;
            else {
                checkNode(node->topLeft, currentDepth + 1);
                checkNode(node->topRight, currentDepth + 1);
                checkNode(node->bottomLeft, currentDepth + 1);
                checkNode(node->bottomRight, currentDepth + 1);
            }
        };

    checkNode(root, 0);
    if (!hasNonLeafAtThisDepth) {
        std::cout << "Tidak ada node non-leaf pada depth " << depth << ",
menghentikan frame di sini." << std::endl;
        break;
    }

    std::cout << "Membuat frame untuk kedalaman " << depth << "..." <<
std::endl;
    FIBITMAP* frameBitmap = createFrameAtDepth(root, width, height,
depth);
    if (!frameBitmap) {
        std::cerr << "Error: Gagal membuat frame untuk kedalaman " <<
depth << std::endl;
        continue;
    }

    std::string frameFile = "frame_" + std::to_string(depth) + ".png";
    frameFileNames.push_back(frameFile);

    if (!FreeImage_Save(FIF_PNG, frameBitmap, frameFile.c_str(), 0)) {
        std::cerr << "Error: Gagal menyimpan frame " << depth <<
std::endl;
    }

    FreeImage_Unload(frameBitmap);
}

std::ostringstream cmd;

if (useMagickExe) {

```

```

        cmd << "magick.exe -delay 50 -loop 0";
    } else {
        cmd << "convert -delay 50 -loop 0";
    }

    for (const auto& frame : frameFileNames) {
        if (fileExists(frame)) {
            cmd << " " << frame;
        } else {
            std::cerr << "Peringatan: File frame " << frame << " tidak
ditemukan." << std::endl;
        }
    }

    cmd << " " << outputPath;

    std::cout << "Menjalankan perintah: " << cmd.str() << std::endl;
    int ret = system(cmd.str().c_str());

    if (ret != 0) {
        std::cerr << "Error: Gagal membuat GIF menggunakan ImageMagick. Kode
error: " << ret << std::endl;
    } else {
        std::cout << "GIF berhasil dibuat!" << std::endl;
    }

    for (const auto& frame : frameFileNames) {
        if (fileExists(frame)) {
            if (std::remove(frame.c_str()) != 0) {
                std::cerr << "Error: Gagal menghapus frame sementara " <<
frame << std::endl;
            }
        }
    }
}

return (ret == 0);
}

```

2.2.5. main.cpp

```

#include <iostream>
#include <string>
#include <chrono>
#include <algorithm>
#include "quadtree.h"

using namespace std;

int main() {
    FreeImage_Initialise();

```

```

FreeImage_SetOutputMessage([](FREE_IMAGE_FORMAT fif, const char *message)
{
    cout << "FreeImage Error";
    if (fif != FIF_UNKNOWN) {
        cout << " (" << FreeImage_GetFormatFromFIF(fif) << ")";
    }
    cout << ": " << message << endl;
});

string inputPath, outputPath, gifPath;
int method;
double threshold;
int minBlockSize;
double targetCompression;

cout << "==== KOMPRESI GAMBAR DENGAN METODE QUADTREE =====< endl;
cout << "Masukkan alamat absolut gambar yang akan dikompresi: ";
getline(cin, inputPath);

cout << "==== Metode Perhitungan Error =====< endl;
cout << "1. Variance" << endl;
cout << "2. MAD" << endl;
cout << "3. Max Pixel Difference" << endl;
cout << "4. SSIM (Bonus)" << endl;
cout << "Masukkan metode perhitungan error: ";
cin >> method;

cout << "Masukkan ambang batas (threshold): ";
cin >> threshold;

cout << "Masukkan ukuran blok minimum: ";
cin >> minBlockSize;

cout << "Masukkan target persentase kompresi (0 untuk menonaktifkan): ";
cin >> targetCompression;

cin.ignore();

cout << "Masukkan alamat absolut gambar hasil kompresi: ";
getline(cin, outputPath);

cout << "Masukkan alamat absolut gif (kosongkan untuk melewati): ";
getline(cin, gifPath);

if (method < 1 || method > 5) {
    cout << "Metode perhitungan error tidak valid! Menggunakan metode
default (Variance)." << endl;
    method = 0;
}

```

```

    if (minBlockSize <= 0) {
        cout << "Ukuran blok minimum tidak valid! Menggunakan nilai default
(4)." << endl;
        minBlockSize = 4;
    }

    // Deteksi format gambar input
    FREE_IMAGE_FORMAT inputFormat = FreeImage_GetFileType(inputPath.c_str());
    if (inputFormat == FIF_UNKNOWN) {
        inputFormat = FreeImage_GetFIFFromFilename(inputPath.c_str());
    }

    if (inputFormat == FIF_UNKNOWN ||
!FreeImage_FIFSupportsReading(inputFormat)) {
        cout << "Format gambar input tidak didukung atau file tidak
ditemukan!" << endl;
        FreeImage_DeInitialise();
        return 1;
    }

    // Load gambar input
    FIBITMAP* originalImage = FreeImage_Load(inputFormat, inputPath.c_str());
    if (!originalImage) {
        cout << "Gagal memuat gambar input!" << endl;
        FreeImage_DeInitialise();
        return 1;
    }

    FIBITMAP* image = FreeImage_ConvertTo24Bits(originalImage);
    FreeImage_Unload(originalImage);

    // Dapatkan dimensi gambar
    int width = FreeImage_GetWidth(image);
    int height = FreeImage_GetHeight(image);

    cout << "\nMemproses gambar " << width << "x" << height << " pixel..." <<
endl;

    auto startTime = chrono::high_resolution_clock::now();

    // Jika target persentase kompresi diaktifkan, temukan threshold optimal
    if (targetCompression > 0) {
        cout << "Mencari threshold optimal untuk target persentase kompresi "
<< targetCompression << "%..." << endl;
        threshold = findThresholdForTargetCompression(image, minBlockSize,
method, targetCompression);
        cout << "Menggunakan threshold optimal: " << threshold << endl;
    }

    cout << "Membangun quadtree..." << endl;

```

```

    QuadTreeNode* root = buildQuadTree(image, 0, 0, width, height,
minBlockSize, threshold, method);

    // Buat gambar output
    FIBITMAP* outputImage = FreeImage_Allocate(width, height, 24);
    if (!outputImage) {
        cout << "Gagal membuat gambar output!" << endl;
        FreeImage_Unload(image);
        delete root;
        FreeImage_DeInitialise();
        return 1;
    }

    cout << "Menggambar hasil kompresi..." << endl;
    drawQuadTree(outputImage, root);

    // Hitung statistik quadtree
    int nodeCount = 0;
    int maxDepth = 0;
    calculateQuadTreeStats(root, nodeCount, maxDepth);

    // Hitung ukuran dan persentase kompresi
    DWORD originalSize = static_cast<DWORD>(width) * height * 3; // 3 bytes
per pixel untuk RGB
    DWORD compressedSize = nodeCount * (sizeof(int) * 4 + sizeof(RGBQUAD) +
sizeof(bool));
    double compressionPercentage = calculateCompressionPercentage(image,
nodeCount);

    auto endTime = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::milliseconds>(endTime -
startTime).count();

    // Simpan gambar output
    FREE_IMAGE_FORMAT outputFormat =
FreeImage_GetFIFFromFilename(outputPath.c_str());
    if (outputFormat == FIF_UNKNOWN) {
        outputFormat = FIF_PNG;
    }

    if (!FreeImage_FIFSupportsWriting(outputFormat)) {
        cout << "Format output tidak didukung untuk penyimpanan. Menggunakan
PNG sebagai gantinya." << endl;
        outputFormat = FIF_PNG;
    }

    bool saveSuccess = FreeImage_Save(outputFormat, outputImage,
outputPath.c_str());
    if (!saveSuccess) {
        cout << "Gagal menyimpan gambar output!" << endl;

```



```

    } else {
        cout << "Gambar hasil kompresi berhasil disimpan ke: " << outputPath
<< endl;
    }

    // Tampilan statistik
    cout << "\n===== STATISTIK KOMPRESI =====" << endl;
    cout << "Waktu eksekusi: " << duration << " ms" << endl;
    cout << "Ukuran gambar sebelum: " << originalSize << " bytes" << endl;
    cout << "Ukuran gambar setelah: " << compressedSize << " bytes" << endl;
    cout << "Persentase kompresi: " << compressionPercentage << "%" << endl;
    cout << "Kedalaman pohon: " << maxDepth << endl;
    cout << "Banyak simpul pada pohon: " << nodeCount << endl;

    // Fungsi membuat GIF
    if (!gifPath.empty()) {
        cout << "\nMemulai proses pembuatan GIF..." << endl;

        string ext = gifPath.substr(gifPath.find_last_of(".") + 1);
        if (ext != "gif" && ext != "GIF") {
            cout << "Menambahkan ekstensi .gif ke nama file" << endl;
            gifPath += ".gif";
        }

        std::replace(gifPath.begin(), gifPath.end(), '\\', '/');

        // Untuk Windows
#ifdef _WIN32
        if (system("where magick > nul 2>&1") == 0) {
            cout << "Menggunakan ImageMagick versi 7+..." << endl;
            if (saveGIF(image, root, gifPath, true)) {
                cout << "GIF berhasil disimpan ke: " << gifPath << endl;
            } else {
                cout << "Gagal menyimpan GIF dengan ImageMagick 7+." << endl;
            }
        } else if (system("where convert > nul 2>&1") == 0) {
            cout << "Menggunakan ImageMagick legacy command..." << endl;
            if (saveGIF(image, root, gifPath, false)) {
                cout << "GIF berhasil disimpan ke: " << gifPath << endl;
            } else {
                cout << "Gagal menyimpan GIF dengan ImageMagick legacy
command." << endl;
            }
        } else {
            cout << "ImageMagick tidak ditemukan. Pastikan ImageMagick
terinstal pada sistem Anda." << endl;
            cout << "Anda dapat mengunduh ImageMagick dari:
https://imagemagick.org/script/download.php" << endl;
            cout << "Dan pastikan untuk mencentang 'Add application directory
to your system path' saat instalasi." << endl;
        }
    }
}

```

```

    }
    #else
    // Pendekatan standard untuk Linux/Mac
    if (saveGIF(image, root, gifPath, false)) {
        cout << "GIF berhasil disimpan ke: " << gifPath << endl;
    } else {
        cout << "Gagal menyimpan GIF. Pastikan ImageMagick terinstal pada
sistem Anda." << endl;
        cout << "Anda dapat mengunduh ImageMagick dari:
https://imagemagick.org/script/download.php" << endl;
    }
    #endif
}

FreeImage_Unload(image);
FreeImage_Unload(outputImage);
delete root;

FreeImage_DeInitialise();

return 0;
}

```

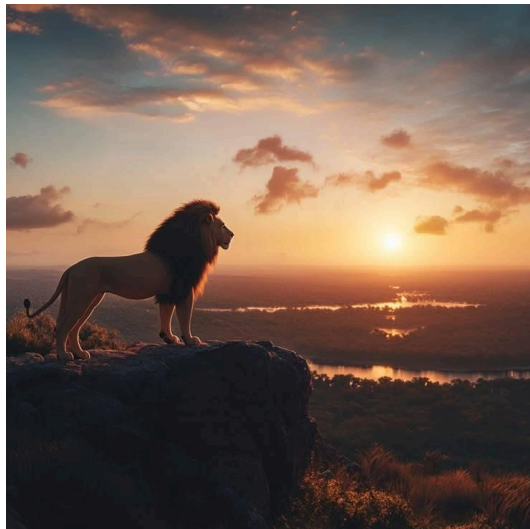
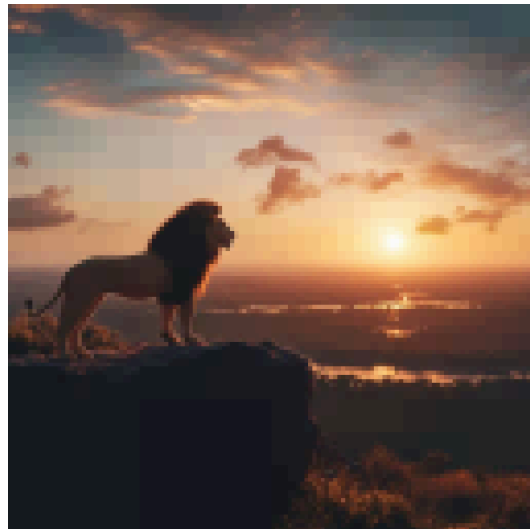
BAB 3

HASIL & ANALISIS PENGUJIAN

3.1. Hasil Pengujian

Berikut merupakan hasil pengujian dengan 8 *test case* yang berbeda:

3.1.1. Test Case 1 (Variance)

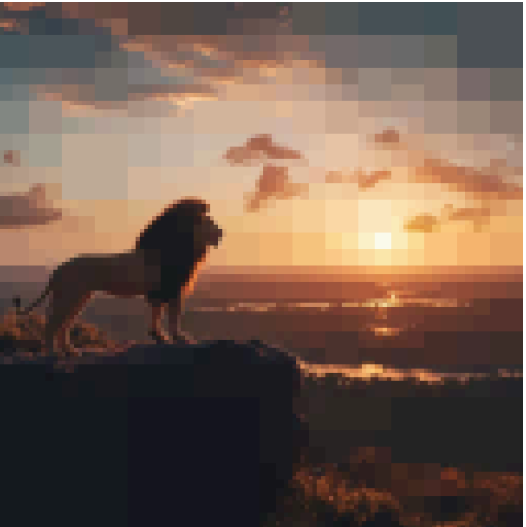
Input	Output
 <pre> PS C:\Users\Raihaan\Tucil2_13523124> .\bin/quadtree_compression ===== KOMPRESI GAMBAR DENGAN METODE QUADTREE ===== Masukkan alamat absolut gambar yang akan dikompresi: test\ori.png ===== Metode Perhitungan Error ===== 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM (Bonus) Masukkan metode perhitungan error: 1 Masukkan ambang batas (threshold): 20 Masukkan ukuran blok minimum: 4 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\variance1.png Masukkan alamat absolut gif (kosongkan untuk melewati): </pre> <p>Masukkan metode perhitungan error: 1 Masukkan ambang batas (threshold): 20 Masukkan ukuran blok minimum: 4 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0</p>	 <pre> Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\variance1.png ===== STATISTIK KOMPRESI ===== Waktu eksekusi: 102 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 205905 bytes Persentase kompresi: 87.3296% Kedalaman pohon: 7 Banyak simpul pada pohon: 9805 PS C:\Users\Raihaan\Tucil2_13523124> </pre> <p>Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\variance1.png</p> <p>===== STATISTIK KOMPRESI ===== Waktu eksekusi: 102 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 205905 bytes Persentase kompresi: 87.3296% Kedalaman pohon: 7 Banyak simpul pada pohon: 9805</p>

3.1.2. Test Case 2 (MAD)

Input	Output
 <pre> PS C:\Users\Raihaan\Tucil2_13523124> ./bin/quadtree_compression ===== KOMPRESI GAMBAR DENGAN METODE QUADTREE ===== Masukkan alamat absolut gambar yang akan dikompresi: test\ori.png ===== Metode Perhitungan Error ===== 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM (Bonus) Masukkan metode perhitungan error: 2 Masukkan ambang batas (threshold): 15 Masukkan ukuran blok minimum: 4 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\MAD1.png Masukkan alamat absolut gif (kosongkan untuk melewati): </pre> <p>Masukkan metode perhitungan error: 2 Masukkan ambang batas (threshold): 15 Masukkan ukuran blok minimum: 4 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\MAD1.png Masukkan alamat absolut gif (kosongkan untuk melewati):</p>	 <pre> Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\MAD1.png ===== STATISTIK KOMPRESI ===== Waktu eksekusi: 74 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 24129 bytes Persentase kompresi: 98.5152% Kedalaman pohon: 7 Banyak simpul pada pohon: 1149 PS C:\Users\Raihaan\Tucil2_13523124> </pre> <p>Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\MAD1.png</p> <p>===== STATISTIK KOMPRESI ===== Waktu eksekusi: 74 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 24129 bytes Persentase kompresi: 98.5152% Kedalaman pohon: 7 Banyak simpul pada pohon: 1149</p>

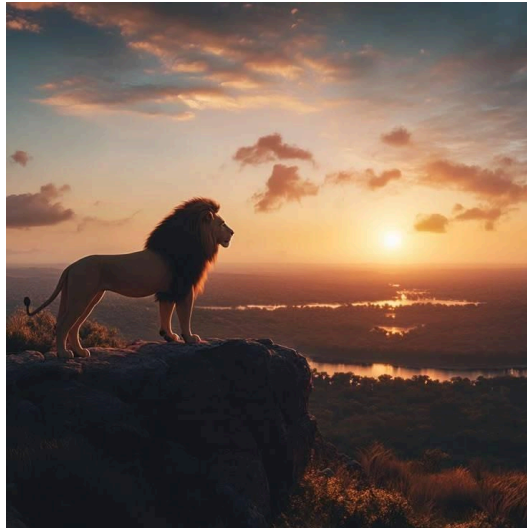
3.1.3. Test Case 3 (Max Pixel Difference)

Input	Output
-------	--------

 <pre> PS C:\Users\Raihaan\Tucil2_13523124> ./bin/quadtree_compression ===== KOMPRESI GAMBAR DENGAN METODE QUADTREE ===== Masukkan alamat absolut gambar yang akan dikompresi: test\ori.png ===== Metode Perhitungan Error ===== 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM (Bonus) Masukkan metode perhitungan error: 3 Masukkan ambang batas (threshold): 50 Masukkan ukuran blok minimum: 4 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\maxpixdif.png Masukkan alamat absolut gif (kosongkan untuk melewati): </pre> <p>Masukkan metode perhitungan error: 3 Masukkan ambang batas (threshold): 50 Masukkan ukuran blok minimum: 4 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\maxpixdif.png Masukkan alamat absolut gif (kosongkan untuk melewati):</p>	 <pre> Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\maxpixdif.png ===== STATISTIK KOMPRESI ===== Waktu eksekusi: 101 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 93009 bytes Persentase kompresi: 94.2767% Kedalaman pohon: 7 Banyak simpul pada pohon: 4429 PS C:\Users\Raihaan\Tucil2_13523124> </pre> <p>Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\maxpixdif.png</p> <p>===== STATISTIK KOMPRESI ===== Waktu eksekusi: 101 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 93009 bytes Persentase kompresi: 94.2767% Kedalaman pohon: 7 Banyak simpul pada pohon: 4429</p>
---	---

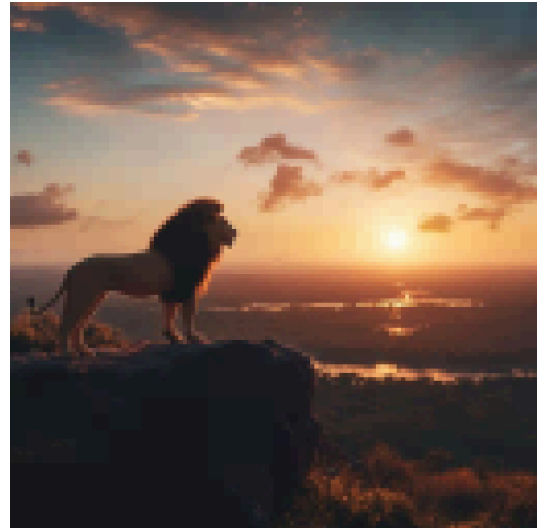
3.1.4. Test Case 4 (Entropy)

Input	Output
-------	--------



```
PS C:\Users\Raihaan\Tucil2_13523124> ./bin/quadtree_compression
===== KOMPRESI GAMBAR DENGAN METODE QUADTREE =====
Masukkan alamat absolut gambar yang akan dikompresi: test\ori.png
===== Metode Perhitungan Error =====
1. Variance
2. MAD
3. Max Pixel Difference
4. Entropy
5. SSIM (Bonus)
Masukkan metode perhitungan error: 4
Masukkan ambang batas (threshold): 3
Masukkan ukuran blok minimum: 4
Masukkan target persentase kompresi (0 untuk menonaktifkan): 0
Masukkan alamat absolut gambar hasil kompresi: test\entropy1.png
Masukkan alamat absolut gif (kosongkan untuk melewati):
```

Masukkan metode perhitungan error: 4
 Masukkan ambang batas (threshold): 3
 Masukkan ukuran blok minimum: 4
 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0
 Masukkan alamat absolut gambar hasil kompresi: test\entropy1.png
 Masukkan alamat absolut gif (kosongkan untuk melewati):



```
Memproses gambar 736x736 pixel...
Membangun quadtree...
Menggambar hasil kompresi...
Gambar hasil kompresi berhasil disimpan ke: test\entropy1.png

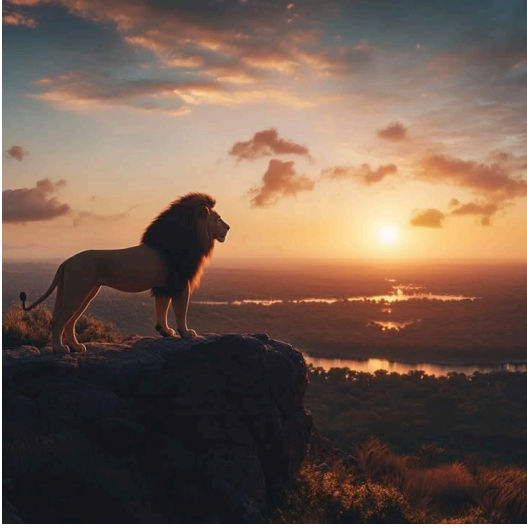

===== STATISTIK KOMPRESI =====
Waktu eksekusi: 141 ms
Ukuran gambar sebelum: 1625088 bytes
Ukuran gambar setelah: 289905 bytes
Persentase kompresi: 82.1607%
Kedalaman pohon: 7
Banyak simpul pada pohon: 13805
PS C:\Users\Raihaan\Tucil2_13523124> █
```

Memproses gambar 736x736 pixel...
 Membangun quadtree...
 Menggambar hasil kompresi...
 Gambar hasil kompresi berhasil
 disimpan ke: test\entropy1.png

===== STATISTIK KOMPRESI =====
 Waktu eksekusi: 141 ms
 Ukuran gambar sebelum: 1625088 bytes
 Ukuran gambar setelah: 289905 bytes
 Persentase kompresi: 82.1607%
 Kedalaman pohon: 7
 Banyak simpul pada pohon: 13805



3.1.5. Test Case 5 (SSIM)

Input	Output
-------	--------

 <pre> PS C:\Users\Raihaan\Tuc112_13523124> ./bin/quadtree_compression ===== KOMPRESI GAMBAR DENGAN METODE QUADTREE ===== Masukkan alamat absolut gambar yang akan dikompresi: test\ori.png ===== Metode Perhitungan Error ===== 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM (Bonus) Masukkan metode perhitungan error: 5 Masukkan ambang batas (threshold): 2000 Masukkan ukuran blok minimum: 4 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\ssim1.png Masukkan alamat absolut gif (kosongkan untuk melewati): </pre> <p>Masukkan metode perhitungan error: 5 Masukkan ambang batas (threshold): 2000 Masukkan ukuran blok minimum: 4 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\ssim1.png Masukkan alamat absolut gif (kosongkan untuk melewati):</p>	 <pre> Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\ssim1.png ===== STATISTIK KOMPRESI ===== Waktu eksekusi: 486 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 222285 bytes Persentase kompresi: 86.3217% Kedalaman pohon: 7 Banyak simpul pada pohon: 10585 </pre> <p>Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\ssim1.png</p> <p>===== STATISTIK KOMPRESI ===== Waktu eksekusi: 486 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 222285 bytes Persentase kompresi: 86.3217% Kedalaman pohon: 7 Banyak simpul pada pohon: 10585</p>
---	---

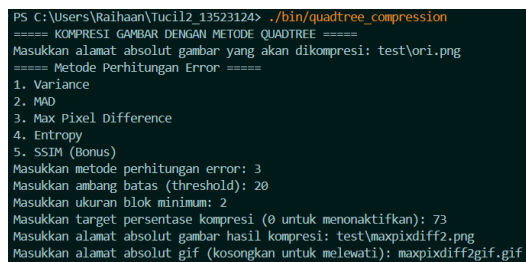
3.1.6. Test Case 6 (MAD 2)

Input	Output
-------	--------

 <pre> PS C:\Users\Raihaan\Tucil2_13523124> ./bin/quadtree_compression ===== KOMPRESI GAMBAR DENGAN METODE QUADTREE ===== Masukkan alamat absolut gambar yang akan dikompresi: test\ori.png ===== Metode Perhitungan Error ===== 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM (Bonus) Masukkan metode perhitungan error: 2 Masukkan ambang batas (threshold): 2 Masukkan ukuran blok minimum: 2 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\MAD2.png Masukkan alamat absolut gif (kosongkan untuk melewati): </pre> <p>Masukkan metode perhitungan error: 2 Masukkan ambang batas (threshold): 2 Masukkan ukuran blok minimum: 2 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\MAD2.png Masukkan alamat absolut gif (kosongkan untuk melewati):</p>	 <pre> Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\MAD2.png ===== STATISTIK KOMPRESI ===== Waktu eksekusi: 113 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 854637 bytes Persentase kompresi: 47.4098% Kedalaman pohon: 8 Banyak simpul pada pohon: 40697 PS C:\Users\Raihaan\Tucil2_13523124> </pre> <p>Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\MAD2.png</p> <p>===== STATISTIK KOMPRESI ===== Waktu eksekusi: 113 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 854637 bytes Persentase kompresi: 47.4098% Kedalaman pohon: 8 Banyak simpul pada pohon: 40697</p>
---	--

3.1.7. Test Case 7 (Manual Kompresi + GIF)

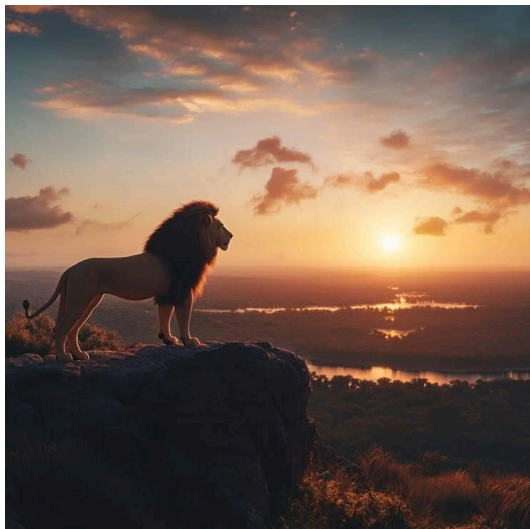
Input	Output
-------	--------

[illegible]

===== STATISTIK KOMPRESI =====
Waktu eksekusi: 979 ms
Ukuran gambar sebelum: 1625088 bytes

	<p>Ukuran gambar setelah: 434889 bytes Persentase kompresi: 73.239% Kedalaman pohon: 8 Banyak simpul pada pohon: 20709</p> <p>Memulai proses pembuatan GIF... Menggunakan ImageMagick versi 7+... Kedalaman pohon quadtree: 8 Membuat frame untuk kedalaman 0... Membuat frame untuk kedalaman 1... Membuat frame untuk kedalaman 2... Membuat frame untuk kedalaman 3... Membuat frame untuk kedalaman 4... Membuat frame untuk kedalaman 5... Membuat frame untuk kedalaman 6... Membuat frame untuk kedalaman 7... Tidak ada node non-leaf pada depth 8, menghentikan frame di sini. Menjalankan perintah: magick.exe -delay 50 -loop 0 frame_0.png frame_1.png frame_2.png frame_3.png frame_4.png frame_5.png frame_6.png frame_7.png maxpixmapdiff2gif.gif GIF berhasil dibuat! GIF berhasil disimpan ke: maxpixmapdiff2gif.gif</p>
--	--

3.1.8. Test Case 8 (Variance 2)

Input	Output
	

<pre> PS C:\Users\Raihaan\Tucil2_13523124> ./bin/quadtree_compression ===== KOMPRESI GAMBAR DENGAN METODE QUADTREE ===== Masukkan alamat absolut gambar yang akan dikompresi: test\ori.png ===== Metode Perhitungan Error ===== 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM (Bonus) Masukkan metode perhitungan error: 3 Masukkan ambang batas (threshold): 10 Masukkan ukuran blok minimum: 1000 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\var1.png Masukkan alamat absolut gif (kosongkan untuk melewati): </pre>	<pre> Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\var1.png ===== STATISTIK KOMPRESI ===== Waktu eksekusi: 30 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 21 bytes Persentase kompresi: 99.9987% Kedalaman pohon: 0 Banyak simpul pada pohon: 1 PS C:\Users\Raihaan\Tucil2_13523124> </pre>
<p>Masukkan metode perhitungan error: 3 Masukkan ambang batas (threshold): 10 Masukkan ukuran blok minimum: 1000 Masukkan target persentase kompresi (0 untuk menonaktifkan): 0 Masukkan alamat absolut gambar hasil kompresi: test\var1.png Masukkan alamat absolut gif (kosongkan untuk melewati):</p>	<p>Memproses gambar 736x736 pixel... Membangun quadtree... Menggambar hasil kompresi... Gambar hasil kompresi berhasil disimpan ke: test\var1.png</p> <p>===== STATISTIK KOMPRESI ===== Waktu eksekusi: 30 ms Ukuran gambar sebelum: 1625088 bytes Ukuran gambar setelah: 21 bytes Persentase kompresi: 99.9987% Kedalaman pohon: 0 Banyak simpul pada pohon: 1</p>

3.2. Analisis Pengujian

Berdasarkan hasil pengujian pada test case 3.1.1 hingga 3.1.8, dapat dianalisis beberapa aspek penting dari program kompresi gambar Quadtree:

Hasil pengujian menunjukkan bahwa metode pengukuran error yang berbeda menghasilkan tingkat kompresi dan kualitas yang bervariasi. Metode MAD memberikan kompresi tertinggi (98.52%) dengan waktu proses tercepat (74 ms), sementara Entropy mempertahankan detail lebih baik namun dengan kompresi yang lebih rendah (82.16%). SSIM, meskipun mengukur error berdasarkan persepsi visual manusia, membutuhkan waktu komputasi yang lebih lama (486 ms).

Ukuran blok minimum memiliki pengaruh signifikan terhadap hasil kompresi. Perbandingan antara ukuran blok minimum 4 dan 2 pada metode MAD menunjukkan perbedaan drastis pada tingkat kompresi (98.52% vs 47.41%) dan jumlah node (1,149 vs 40,697). Ini memperlihatkan trade-off antara tingkat kompresi dan preservasi detail (yang mana menjadi karakteristik yang sesuai dengan algoritma kompresi gambar Quadtree berbasis divide and conquer).

Implementasi fitur target persentase kompresi bekerja dengan baik, mampu menyesuaikan threshold secara otomatis untuk mencapai kompresi yang mendekati target. Pada test case 3.1.7, program berhasil mencapai kompresi 73.24% dari target 73% dengan threshold 21.0938.

Kasus ekstrem pada test case 3.1.8 dengan ukuran blok minimum 1000 menghasilkan satu simpul dengan kompresi 99.99%, menunjukkan bahwa algoritma bekerja sesuai ekspektasi bahkan dalam kondisi parameter yang tidak biasa.

Dari perspektif kompleksitas algoritma, implementasi divide and conquer pada Quadtree menunjukkan kompleksitas waktu $O(n \log n)$ untuk n piksel, dengan waktu eksekusi yang dipengaruhi oleh kompleksitas gambar, nilai threshold, metode pengukuran error, dan ukuran blok minimum.

Visualisasi GIF dari proses pembentukan Quadtree berhasil menghasilkan representasi yang jelas tentang bagaimana algoritma divide and conquer secara progresif membagi gambar menjadi blok-blok yang semakin kecil.

Secara keseluruhan, program kompresi gambar dengan metode Quadtree berhasil mengimplementasikan algoritma divide and conquer dengan efektif, menawarkan fleksibilitas parameter yang memungkinkan penyesuaian antara tingkat kompresi dan kualitas visual sesuai kebutuhan.

BAB 4

KESIMPULAN

4.1 Kesimpulan

Berdasarkan pengembangan dan pengujian program kompresi gambar dengan metode Quadtree, dapat disimpulkan bahwa algoritma divide and conquer terbukti efektif untuk implementasi kompresi gambar berbasis Quadtree. Program berhasil mengimplementasikan empat metode pengukuran error wajib (Variance, MAD, Max Pixel Difference, dan Entropy) dengan karakteristik berbeda, di mana MAD memberikan kompresi tertinggi dan Entropy mempertahankan detail lebih baik. Parameter seperti threshold, ukuran blok minimum, dan metode pengukuran error memberikan fleksibilitas dalam menyeimbangkan trade-off antara tingkat kompresi dan kualitas gambar. Fitur target persentase kompresi secara otomatis mencari threshold optimal untuk mencapai kompresi yang diinginkan, menyederhanakan proses bagi pengguna. Dengan kompleksitas waktu $O(n \log n)$ untuk n piksel, pendekatan divide and conquer menunjukkan efisiensi komputasional yang baik. Visualisasi GIF dari proses pembentukan Quadtree memberikan pemahaman jelas tentang cara kerja algoritma, sementara implementasi SSIM memberikan pendekatan yang lebih sesuai dengan persepsi visual manusia. Secara keseluruhan, program ini berhasil mendemonstrasikan penerapan konsep divide and conquer dalam pemrosesan gambar dengan menyediakan kompresi yang efektif, fleksibel, dan adaptif terhadap berbagai kebutuhan.

BAB 5

BONUS, REPOSITORY, TABEL EVALUASI

5.1. Bonus yang Diimplementasikan

Program ini mencakup semua bonus yang diimplementasikan

5.1.1. Target Persentasi Kompresi

Fungsi `findThresholdForTargetCompression()` mengimplementasikan algoritma binary search untuk mencari threshold optimal yang menghasilkan persentase kompresi sedekat mungkin dengan target yang diinginkan pengguna. Algoritma ini bekerja dengan melakukan iterasi pencarian threshold, membangun quadtree, dan menghitung persentase kompresi hingga mendekati target yang diinginkan. Jadi, pengguna bisa memasukkan target kompresi yang diinginkan dan program akan mencari threshold paling optimal dan terdekat dengan target pengguna.

5.1.2. Implementasi Structural Similarity Index (SSIM)

Metode perhitungan error SSIM diimplementasikan dalam fungsi `calculateSSIM()`. Algoritma ini menghitung indeks similaritas struktural dengan membandingkan statistik luminansi, kontras, dan struktur antara blok gambar asli dan blok dengan warna rata-rata.

5.1.3. GIF Proses Kompresi

Fungsi `saveGIF()` membuat visualisasi proses pembentukan Quadtree dalam format GIF. Algoritma ini menghasilkan frame untuk setiap kedalaman pohon Quadtree, kemudian menggunakan library eksternal ImageMagick untuk menggabungkan frame-frame tersebut menjadi sebuah animasi GIF yang menunjukkan progresif pembentukan struktur kompresi.

5.2. Pranala ke Repository

Link Repository GitHub : https://github.com/fliegenhaan/Tucil2_13523124

LAMPIRAN

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	✓	
4	Mengimplementasi seluruh metode perhitungan error wajib	✓	
5	[Bonus] Implementasi persentase kompresi sebagai parameter tambahan	✓	
6	[Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	✓	
7	[Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	✓	
8	Program dan laporan dibuat (kelompok) sendiri	✓	