

LAPORAN TUGAS KECIL II
IF2211 - STRATEGI ALGORITMA
Kompresi Gambar Dengan Metode Quadtree



Disusun oleh :
Muhammad Alfansya - 13523005
Muhammad Raihan Nazhim Oktana – 13523021

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

Daftar Isi

Daftar Isi.....	2
BAB I DESKRIPSI PROGRAM.....	3
1.1. Quadtree.....	3
1.2. Deskripsi Tugas.....	5
BAB II ALGORITMA DIVIDE AND CONQUER.....	7
2.1. Algoritma Divide and Conquer.....	7
2.1.1. Divide.....	7
2.1.2. Conquer.....	8
2.1.3. Combine.....	14
BAB III HASIL ANALISIS.....	16
3.1. Analisis Percobaan.....	16
3.2. Analisis Kompleksitas Program.....	16
BAB IV PENJELASAN IMPLEMENTASI BONUS.....	17
4.1. Implementasi Bonus Target Compression.....	17
4.2. Analisis Kompleksitas Bonus Target Compression.....	19
BAB V SOURCE CODE.....	20
5.1. Program main.cpp.....	20
5.2. Program quadtree.hpp.....	26
5.3. Program quadtree.cpp.....	28
5.4. Program image_util.hpp.....	35
5.5. Program image_util.cpp.....	36
BAB VI TEST CASE.....	39
6.1. Kumpulan Test Case Hasil Gambar Optimal.....	39
6.1.1. Menggunakan Metode 1.....	39
6.1.2. Menggunakan Metode 2.....	39
6.1.3. Menggunakan Metode 3.....	40
6.1.4. Menggunakan Metode 4.....	40
6.2. Kumpulan Test Case Hasil Kompresi Optimal.....	41
6.2.1. Menggunakan Metode 1.....	41
6.2.2. Menggunakan Metode 2.....	41
6.2.3. Menggunakan Metode 3.....	42
6.2.4. Menggunakan Metode 4.....	42
6.3. Kumpulan Test Case Lainnya.....	43
6.3.1. Hasil Gambar Bonus Target Compression.....	43
6.3.2. Hasil Gambar Hanya 1 Potongan.....	43
6.3.3. Error Handling 1 : Format File Tidak Tepat / File Tidak Tersedia.....	44
6.3.4. Error Handling 2 : Pilihan Metode Tidak Valid.....	44
6.3.5. Error Handling 3 : Nilai Threshold Tidak Valid.....	44
6.3.6. Error Handling 4 : Minimum Block Size Tidak Valid.....	44
6.3.7. Error Handling 5 : Target Compression Tidak Valid.....	44
LAMPIRAN.....	45

BAB I

DESKRIPSI PROGRAM

1.1. Quadtree



Gambar 1. Quadtree dalam Kompresi Gambar

(Sumber: <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>)

Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis sistem warna RGB, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika bagian tersebut tidak seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.

Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

Tabel 1. Metode Pengukuran Error

Metode	Formula
<u>Variance</u>	$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
Mean Absolute Deviation (MAD)	$MAD_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $
	$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$
	MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
	N = Banyaknya piksel dalam satu blok

Max Pixel Difference	$D_c = \max(P_{i,c}) - \min(P_{i,c})$
	$D_{RGB} = \frac{D_R + D_G + D_B}{3}$
	$D_c = \text{Selisih antara piksel dengan nilai max dan min tiap kanal warna } c \text{ (R, G, B) dalam satu blok}$ $P_{i,c} = \text{Nilai piksel pada posisi } i \text{ untuk channel warna } c$
Entropy	$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$
	$H_{RGB} = \frac{H_R + H_G + H_B}{3}$
	$H_c = \text{Nilai entropi tiap kanal warna } c \text{ (R, G, B) dalam satu blok}$ $P_c(i) = \text{Probabilitas piksel dengan nilai } i \text{ dalam satu blok untuk tiap kanal warna } c \text{ (R, G, B)}$
[Bonus] <u>Structural Similarity Index (SSIM)</u> <u>(Referensi tambahan)</u>	$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$
	$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$
	Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Silakan lakukan eksplorasi untuk memahami serta memperoleh nilai konstanta pada formula SSIM, asumsikan gambar yang akan diuji adalah 24-bit RGB dengan 8-bit per kanal.

1.2. Deskripsi Tugas

Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (Divide and Conquer):

1. Inisialisasi dan Persiapan Data

Masukkan gambar yang akan dikompresi akan diolah dalam format matriks piksel dengan nilai intensitas berdasarkan sistem warna RGB. Berikut adalah parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan kompresi gambar:

- Metode perhitungan variansi: pilih metode perhitungan variansi berdasarkan opsi yang tersedia pada Tabel 1 (di spesifikasi).
 - Threshold variansi: nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.
 - Minimum block size: ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.
2. Perhitungan Error
- Untuk setiap blok gambar yang sedang diproses, hitung nilai variansi menggunakan metode yang dipilih sesuai Tabel 1 (di spesifikasi).
3. Pembagian Blok
- Bandingkan nilai variansi blok dengan threshold:
- Jika variansi di atas threshold (cek kasus khusus untuk metode bonus), ukuran blok lebih besar dari minimum block size, dan ukuran blok setelah dibagi menjadi empat tidak kurang dari minimum block size, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.
 - Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.
4. Normalisasi Warna
- Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai dengan rata-rata nilai RGB blok.
5. Rekursi dan Penghentian
- Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua blok memenuhi salah satu dari dua kondisi berikut:
- Error blok berada di bawah threshold.
 - Ukuran blok setelah dibagi menjadi empat kurang dari minimum block size.
6. Penyimpanan dan Output

Rekonstruksi gambar dilakukan berdasarkan struktur QuadTree yang telah dihasilkan selama proses kompresi. Gambar hasil rekonstruksi akan disimpan sebagai file terkompresi. Selain itu, persentase kompresi akan dihitung dan disertakan dengan rumus sesuai dengan yang terlampir pada dokumen ini. Persentase kompresi ini memberikan gambaran mengenai efisiensi metode kompresi yang digunakan.

BAB II

ALGORITMA DIVIDE AND CONQUER

2.1. Algoritma Divide and Conquer

Algoritma Divide and Conquer adalah algoritma pemecahan masalah dengan cara membagi masalah besar menjadi beberapa submasalah yang lebih kecil dan serupa, menyelesaikan masing-masing submasalah secara rekursif, lalu menggabungkan hasilnya untuk membentuk solusi akhir. Langkah-langkah utamanya terdiri dari: *Divide* (membagi masalah), *Conquer* (menyelesaikan submasalah), dan *Combine* (menggabungkan solusi submasalah). Salah satu penerapan divide and conquer adalah penerapan dalam quadtree, suatu masalah akan dipecah menjadi node-node tree yang lebih kecil hingga ke kasus basis nya.

Pada program Kompresi Gambar Dengan Metode Quadtree melalui algoritma divide and conquer diterapkan saat memecah gambar dan membentuk node-node quadtree, penghitungan error, dan reconstruct tree.

2.1.1. Divide

Gambar diperlakukan sebagai kumpulan pixel yang kemudian pixel-pixel tersebut dibagi menjadi kumpulan node-node pixel yang lebih kecil secara rekursif hingga menemui kasus basis, kasus basis adalah parameter-parameter yang diinput user, yaitu threshold variansi dan/atau minimum block size.

```

function BuildQuadTree(image, startX, startY, blockWidth,
    blockHeight, minBlockSize, threshold, method) -> QuadNode

KAMUS LOKAL

type Pixel :
    r, g, b : float

type QuadNode :
    avg_color : pixel
    startX , startY : integer
    blockw , blockh : integer
    child : array [0..3] of QuadNode
    avgColor : Pixel
    newNode : QuadNode

ALGORITMA
    newNode.startX <- startX

```

```

newNode.startY <- startY
newNode.blockWidth <- blockWidth
newNode.blockHeight <- blockHeight

errorValue <- ComputeBlockError(image, startX, startY,
                                 blockWidth, blockHeight, method, avgColor)
newNode.avgColor <- avgColor

if (errorValue > threshold AND blockWidth > minBlockSize and
     blockHeight > minBlockSize) then:
    halfWidth <- floor(blockWidth / 2)
    halfHeight <- floor(blockHeight / 2)

    newNode.child[0] <- BuildQuadTree(image, startX, startY,
                                         halfWidth, halfHeight, minBlockSize, threshold, method)
    newNode.child[1] <- BuildQuadTree(image, startX +
                                         halfWidth, startY, blockWidth - halfWidth, halfHeight,
                                         minBlockSize, threshold, method)
    newNode.child[2] <- BuildQuadTree(image, startX, startY +
                                         halfHeight, halfWidth, blockHeight - halfHeight, minBlockSize,
                                         threshold, method)
    newNode.child[3] <- BuildQuadTree(image, startX +
                                         halfWidth, startY + halfHeight, blockWidth - halfWidth,
                                         blockHeight - halfHeight, minBlockSize, threshold, method)

-> newNode

```

2.1.2. Conquer

Node-node pixel yang sudah dipecah akan diproses perhitungan error-nya. Tersedia 4 pilihan metode pengukuran error, yaitu variance, mean absolute deviation (MAD), max pixel difference, dan entropy. Program akan menggunakan metode pilihan user. Node-node pixel akan mengalami normalisasi warna sesuai nilai rata-rata RGB blok sekitarnya.

```

function ComputeVariance(image, startX, startY,
                         blockWidth, blockHeight, average) -> float

```

KAMUS LOKAL

```

type Pixel :
    r, g, b : float
    sumR, sumG, sumB : integer
    n : integer
    meanR, meanG, meanB : float
    varR, varG, varB : float
    i, j : integer
    dr, dg, db : float

```

ALGORITMA

```

sumR <- 0 , sumG <- 0, sumB <- 0
varR <- 0 , varG <- 0, varB <- 0
n ← blockWidth * blockHeight

for (i ← startY TO startY + blockHeight - 1) do
    for (j ← startX TO startX + blockWidth - 1) do
        sumR ← sumR + image[i][j].r
        sumG ← sumG + image[i][j].g
        sumB ← sumB + image[i][j].b
    End for
End for

meanR ← sumR / n
meanG ← sumG / n
meanB ← sumB / n

average.r ← round(meanR)
average.g ← round(meanG)
average.b ← round(meanB)

for (i ← startY TO startY + blockHeight - 1) do
    for (j ← startX TO startX + blockWidth - 1) do
        dr ← image[i][j].r - meanR
        dg ← image[i][j].g - meanG
        db ← image[i][j].b - meanB

```

```

        varR ← varR + dr * dr
        varG ← varG + dg * dg
        varB ← varB + db * db
    Endfor
Endfor

varR ← varR / n
varG ← varG / n
varB ← varB / n

-> (varR + varG + varB) / 3.0

```

```

function compute_mad(image, start_x, start_y, block_width,
    block_height, average) -> float

```

KAMUS LOKAL

```

type pixel :
    r, g, b : float
    sum_r, sum_g, sum_b : integer
    n : integer
    mean_r, mean_g, mean_b : float
    mad_r, mad_g, mad_b : float
    i, j : integer

```

ALGORITMA

```

    sum_r <- 0, sum_g <- 0, sum_b <- 0
    mad_r <- 0, mad_g <- 0, mad_b <- 0
    n <- block_width * block_height

    for (i <- start_y to start_y + block_height - 1) do
        for (j <- start_x to start_x + block_width - 1) do
            sum_r <- sum_r + image[i][j].r
            sum_g <- sum_g + image[i][j].g
            sum_b <- sum_b + image[i][j].b
        endfor
    endfor

    mean_r <- sum_r / n

```

```

mean_g <- sum_g / n
mean_b <- sum_b / n

average.r <- round(mean_r)
average.g <- round(mean_g)
average.b <- round(mean_b)

for(i <- start_y to start_y + block_height - 1) do
    for(j <- start_x to start_x + block_width - 1) do
        mad_r <- mad_r + abs(image[i][j].r - mean_r)
        mad_g <- mad_g + abs(image[i][j].g - mean_g)
        mad_b <- mad_b + abs(image[i][j].b - mean_b)
    endfor
endfor

mad_r <- mad_r / n
mad_g <- mad_g / n
mad_b <- mad_b / n

-> (mad_r + mad_g + mad_b) / 3.0

```

```

function compute_max_pixel_diff(image, start_x, start_y,
                                block_width, block_height, average) -> float

```

KAMUS LOKAL

```

type_pixel :
    r, g, b : float
    min_r, min_g, min_b : integer
    max_r, max_g, max_b : integer
    sum_r, sum_g, sum_b : integer
    mean_r, mean_g, mean_b : float
    dr, dg, db : float
    i, j : integer
    n : integer
    r, g, b : integer

```

ALGORITMA

```

min_r <- 255 ; min_g <- 255 ; min_b <- 255

```

```

max_r <- 0 ; max_g <- 0 ; max_b <- 0
sum_r <- 0 ; sum_g <- 0 ; sum_b <- 0
n <- block_width * block_height

for (i <- start_y to start_y + block_height - 1) do
  for (j <- start_x to start_x + block_width - 1) do
    r <- image[i][j].r
    g <- image[i][j].g
    b <- image[i][j].b

    if (r > max_r) then max_r <- r
    if (r < min_r) then min_r <- r
    if (g > max_g) then max_g <- g
    if (g < min_g) then min_g <- g
    if (b > max_b) then max_b <- b
    if (b < min_b) then min_b <- b

    sum_r <- sum_r + r
    sum_g <- sum_g + g
    sum_b <- sum_b + b
  endfor
endfor

mean_r <- sum_r / n
mean_g <- sum_g / n
mean_b <- sum_b / n

average.r <- round(mean_r)
average.g <- round(mean_g)
average.b <- round(mean_b)

dr <- max_r - min_r
dg <- max_g - min_g
db <- max_b - min_b

-> (dr + dg + db) / 3.0

```

```

function compute_entropy(image, start_x, start_y, block_width,
    block_height, average) -> float

KAMUS LOKAL
    type pixel :
        r, g, b : float
        sum_r, sum_g, sum_b : integer
        mean_r, mean_g, mean_b : float
        i, j, n : integer
        vhist_r, vhist_g, vhist_b : array[0..255] of integer
        er, eg, eb : float
        p : float

ALGORITMA
    sum_r <- 0 , sum_g <- 0 , sum_b <- 0
    n <- block_width * block_height

    for (i <- start_y to start_y + block_height - 1) do
        for (j <- start_x to start_x + block_width - 1) do
            sum_r <- sum_r + image[i][j].r
            sum_g <- sum_g + image[i][j].g
            sum_b <- sum_b + image[i][j].b
        endfor
    endfor

    mean_r <- sum_r / n
    mean_g <- sum_g / n
    mean_b <- sum_b / n

    average.r <- round(mean_r)
    average.g <- round(mean_g)
    average.b <- round(mean_b)

    for (i <- 0 to 255) do
        vhist_r[i] <- 0
        vhist_g[i] <- 0
        vhist_b[i] <- 0
    endfor

```

```

for(i <- start_y to start_y + block_height - 1) do
    for(j <- start_x to start_x + block_width - 1) do
        vhist_r[image[i][j].r] <- vhist_r[image[i][j].r] + 1
        vhist_g[image[i][j].g] <- vhist_g[image[i][j].g] + 1
        vhist_b[image[i][j].b] <- vhist_b[image[i][j].b] + 1
    endfor
endfor

er <- 0 ; eg <- 0 ; eb <- 0

for(i <- 0 to 255) do
    if(vhist_r[i] > 0) then
        p <- vhist_r[i] / n
        er <- er - p * log2(p)
    endif
    if(vhist_g[i] > 0) then
        p <- vhist_g[i] / n
        eg <- eg - p * log2(p)
    endif
    if(vhist_b[i] > 0) then
        p <- vhist_b[i] / n
        eb <- eb - p * log2(p)
    endif
endfor

-> (er + eg + eb) / 3.0

```

2.1.3. Combine

Berikutnya node-node yang sudah diproses akan di reconstruct menjadi gambar utuh yang sudah terkompresi sesuai dengan parameter. Gambar yang di-reconstruct menggunakan average RGB color yang sudah dinormalisasi di proses conquer.

```

procedure ReconstructQuadtree(root , res_img)
KAMUS LOKAL

```

```

Type Pixel :
    r , g , b : float

Type QuadNode :
    avg_color : Pixel
    startx , starty : integer
    blockw , blockh : integer
    child : array [0..3] of QuadNode
    check_leaf : boolean
    i , j : integer

```

ALGORITMA

```

if (root is not null) then
    check_leaf <- true
    for (i <- 0 to 3) do
        if (root.child[i] is not null) then
            check_leaf <- false
            break
        endif
    endfor
    if (check_leaf) then
        for (i <- root.starty to root.starty + root.blockh - 1) do
            for (j <- root.startx to root.startx + root.blockw - 1) do
                res_img[i][j] <- root.avg_color
            endfor
        endfor
    else
        for (i <- 0 to 3) do
            reconstruct_quadtree(root.child[i] , res_img)
        endfor
    endif
endif

```

BAB III

HASIL ANALISIS

3.1. Analisis Percobaan

Gambar dipecah dengan algoritma divide and conquer menjadi lebih kecil membentuk quadtree, setiap node akan memiliki 4 node anak, secara rekursif. Node yang sudah menemui kasus basis sesuai parameter inputan akan berhenti dipecah.

Parameter threshold yaitu batas minimum variansi hasil perhitungan error. Makin tinggi nilai threshold maka hasil kompresi akan semakin besar. Parameter minimal block yaitu batas minimum ukuran block pixel yang dipecah. Makin besar ukuran minimal block maka hasil kompresi akan semakin besar.

Dari hasil percobaan dapat diketahui bahwa hasil kompresi bergantung pada tingkat keberagaman lokal pixel warna pada gambar. Daerah atau bagian gambar yang memiliki pixel beragam atau gambar dengan warna lebih detail akan mengalami pemecahan lebih banyak daripada daerah gambar yang memiliki detail lebih kecil.

3.2. Analisis Kompleksitas Program

1. Divide

Gambar dipecah membentuk quadtree secara rekursif hingga menemui kasus basis. Jika gambar memiliki ukuran $N \times N$ dan tiap perulangan gambar akan dipecah menjadi 4, maka algoritma ini akan memiliki kompleksitas $O(N^2 \cdot \log N)$

2. Conquer

- Metode Variance memiliki kompleksitas $O(N^2)$ karena akan menghitung matriks pixel miliki gambar.
- Metode Mean Absolute Deviation(MAD) memiliki kompleksitas $O(N^2)$ karena program menghitung matriks pixel milik gambar
- Metode Max Pixel Difference memiliki kompleksitas $O(N^2)$ karena harus menentukan nilai maksimal dan minimal tiap matriks.
- Metode Entropy memiliki kompleksitas $O(N^2)$ karena perlu menghitung entropi tiap matriks bagian gambar.

3. Combine

Algoritma ini memanggil dirinya secara rekursif untuk tiap child node yang dimiliki tiap node. Algoritma ini memiliki kompleksitas $O(\log N)$.

BAB IV

PENJELASAN IMPLEMENTASI BONUS

4.1. Implementasi Bonus Target Compression

Subprogram untuk menentukan threshold terbaik untuk menemukan threshold yang sesuai agar persentase kompresi sesuai dengan target dilakukan menggunakan algoritma binary search. Yaitu dengan melakukan penempatan minimum threshold dan maksimum threshold di kedua ujung, lalu secara iterasi mengambil titik tengah untuk lanjut pada step berikutnya. Proses pemampatan kedua pointer menuju titik tujuan ini dilakukan sebanyak k kali yang disepakati sebanyak 20 kali. Jika belum ditemukan yang sesuai dalam galat yang sudah ditentukan (0,05%) atau sampai di ujung, maka program akan dihentikan.

```

procedure FindBestThreshold(input_image , width , height , min_block_size , method ,
                             original_size , target_compression , threshold OUT)

KAMUS LOKAL
  mint , maxt , mid : float
  best_threshold , best_diff : float
  best_final_size , k , temp_size : integer
  temp_root : QuadNode
  temp_image : matrix of Pixel
  ratio , diff , tolerance_ratio : float
  check : boolean

ALGORITMA
  if (method = 1) then
    mint ← 0.0
    maxt ← 10000.0
  else if (method = 2) then
    mint ← 0.0
    maxt ← 255.0
  else if (method = 3) then
    mint ← 0.0
    maxt ← 765.0
  else if (method = 4) then
    mint ← 0.0
    maxt ← 24.0

```

```

else
    mint ← 0.0
    maxt ← 1.0
    best_diff ← ∞
    best_threshold ← threshold
    best_final_size ← 0
    tolerance_ratio ← 0.0005
    for (k ← 0 to 19) do
        mid ← (mint + maxt) / 2
        temp_root ← BuildQuadTree(input_image , 0 , 0 , width , height ,
            min_block_size , mid , method)
        for (i ← 0 to height - 1) do
            for (j ← 0 to width - 1) do
                temp_image[i][j] ← input_image[i][j]
        ReconstructQuadTree(temp_root , temp_image)
        FreeQuadTree(temp_root)
        check ← save_image("temp_image.png" , temp_image , width , height)
        temp_size ← size_of_file("temp_image.png")
        ratio ← abs(1.0 - (temp_size / original_size))
        diff ← abs(ratio - target_compression)
        if (diff < best_diff) then
            best_diff ← diff
            best_threshold ← mid
            best_final_size ← temp_size
        if (ratio < target_compression) then
            mint ← mid
        else
            maxt ← mid
        if ((best_diff < tolerance_ratio) or (abs(maxt - mint) < 0.000001) or (k =
            19)) then
            break
    threshold ← best_threshold

```

4.2. Analisis Kompleksitas Bonus Target Compression

Berdasarkan algoritma diatas, terlihat bahwa program bonus yang dibuat disusun dengan algoritma Binary Search (BS) yang merupakan bagian dari Algoritma Divide and Conquer. Hal ini terlihat dari penyesuaian nilai minimum threshold dan maksimum threshold yang digunakan pada setiap iterasinya. Oleh karena itu, dapat disimpulkan bahwa kompleksitas algoritma yang digunakan dapat dihitung dari banyak langkah yang digunakan dalam setiap iterasi kompresi dikali banyak penyesuaian yang dilakukan. Sehingga, kompleksitas algoritma keseluruhan program bonus ini adalah $O(k * N^2 * \log(N))$, namun khusus ini hanya $O(k)$.

BAB V

SOURCE CODE

5.1. Program main.cpp

```

/* Nama      : Muhammad Raihan Nazhim Oktana & Muhammad Alfansya */
/* NIM       : 13523021 & 13523005 - Teknik Informatika          */
/* Tanggal   : Jumat, 11 April 2025                         */
/* Tugas     : Stima (IF2211-24) - Tucil 2 - Divide & Conquer  */
/* File      : main.cpp                                         */
/* Deskripsi  : F01 - Main - File Main Program CPP           */
/* PJ F01    : Muhammad Alfansya - 13523005                  */

#include <bits/stdc++.h>
#include "quadtree.hpp"
#include "image_util.hpp"
using namespace std;

int main() {
    cout << "===== " << endl;
    cout << " SELAMAT DATANG DI PROGRAM KOMPRESI GAMBAR " << endl;
    cout << "          QUADTREE BY RAIHAN & ALFANSYA          " << endl;
    cout << "          13523021 & 13523005          " << endl;
    cout << "===== " << endl;
    bool check = false;
    string input_path , lower_input;
    int width , height;
    vector<vector<Pixel>> input_image;
    auto str_ends_with = [] (const string& str , const string& sfx) -> bool {
        return str.size() >= sfx.size() && str.compare(str.size() - sfx.size()
, sfx.size() , sfx) == 0;
    };
    cout << "Masukkan path gambar yang akan dikompresi (JPG / JPEG / PNG) : ";
    getline(cin , input_path);
    while (!check) {
        lower_input = input_path;
        transform(lower_input.begin() , lower_input.end() , lower_input.begin()
, ::tolower);

```

```

        while (!(str_ends_with(lower_input, ".jpg") ||
str_ends_with(lower_input, ".jpeg") || str_ends_with(lower_input, ".png"))) {
            cout << "Error : File input harus bertipe .jpg / .jpeg / .png" <<
endl;
            cout << "[RE] Masukkan path gambar yang akan dikompresi (JPG / JPEG
/ PNG) : ";
            getline(cin, input_path);
            lower_input = input_path;
            transform(lower_input.begin(), lower_input.end(), lower_input.begin(),
::tolower);
        }
        check = load_image(input_path, input_image, width, height);
        if (!check) {
            cout << "Error : Gagal load gambar. Pastikan file tersedia, aman,
dan path sesuai." << endl;
            cout << "[RE] Masukkan path gambar yang akan dikompresi (JPG / JPEG
/ PNG) : ";
            getline(cin, input_path);
        }
    }
    cout << "-----" << endl;
    cout << "Pilihan Metode Perhitungan Error :" << endl;
    cout << "1. Variance" << endl;
    cout << "2. Mean Absolute Deviation (MAD)" << endl;
    cout << "3. Max Pixel Difference" << endl;
    cout << "4. Entropy" << endl;
    cout << "5. Structural Similarity Index (SSIM)" << endl;
    cout << "Masukkan metode yang ingin digunakan (angka) : ";
    int method;
    cin >> method;
    while (method < 1 || method > 4) {
        cout << "Error : Metode tidak valid. Hanya boleh 1 - 4 (SSIM belum
tersedia)." << endl;
        cout << "[RE] Masukkan metode yang ingin digunakan (angka) : ";
        cin >> method;
    }
    cout << "-----" << endl;
    if (method == 1) {

```

```
    cout << "Saran Range Threshold Metode Variance : 0 <= ... <= 10000" <<
endl;
} else if (method == 2) {
    cout << "Saran Range Threshold Metode MAD : 0 <= ... <= 255" << endl;
} else if (method == 3) {
    cout << "Saran Range Threshold Metode MaxPixelDiff : 0 <= ... <= 765"
<< endl;
} else if (method == 4) {
    cout << "Saran Range Threshold Metode Entropy : 0 <= ... <= 24" <<
endl;
} else {
    cout << "Saran Range Threshold Metode SSIM : 0 <= ... <= 1" << endl;
}
cout << "Masukkan nilai threshold (ambang batas) : ";
double threshold;
cin >> threshold;
while (threshold < 0) {
    cout << "Error : Threshold tidak valid. Threshold tidak boleh negatif,
usahakan sesuai saran." << endl;
    cout << "[RE] Masukkan nilai threshold (ambang batas) : ";
    cin >> threshold;
}
cout << "-----" << endl;
cout << "Masukkan minimum block size yang diinginkan (panjang sisi) : ";
int min_block_size;
cin >> min_block_size;
while (min_block_size < 1 || min_block_size > min(width , height)) {
    cout << "Error : Ukuran tidak valid. Hanya boleh 1 - " << min(width ,
height) << endl;
    cout << "[RE] Masukkan minimum block size yang diinginkan (panjang
sisi) : ";
    cin >> min_block_size;
}
cout << "-----" << endl;
cout << "Masukkan target persentase kompresi (masukkan 0 untuk
menonaktifkan) : ";
double target_compression;
cin >> target_compression;
```

```
while (target_compression < 0.0 || target_compression > 1.0) {
    cout << "Error : Target kompresi tidak valid. Hanya boleh 0.0 < ... <
1.0." << endl;
    cout << "[RE] Masukkan target persentase kompresi (masukkan 0 untuk
menonaktifkan) : ";
    cin >> target_compression;
}
cin.ignore();
cout << "-----" << endl;
cout << "Masukkan output path gambar hasil kompresi (PNG) : ";
string output_path;
getline(cin , output_path);
lower_input = output_path;
transform(lower_input.begin() , lower_input.end() , lower_input.begin() ,
::tolower);
while (!(str_ends_with(lower_input , ".png"))) {
    cout << "Error : File output harus bertipe .png" << endl;
    cout << "[RE] Masukkan output path gambar hasil kompresi (PNG) : ";
    getline(cin , output_path);
    lower_input = output_path;
    transform(lower_input.begin() , lower_input.end() , lower_input.begin() ,
::tolower);
}
cout << "-----" << endl;
cout << "Masukkan path output GIF (enter untuk menonaktifkan) : ";
string gif_path;
getline(cin , gif_path);
lower_input = gif_path;
transform(lower_input.begin() , lower_input.end() , lower_input.begin() ,
::tolower);
while (!gif_path.empty() && !str_ends_with(lower_input , ".gif")) {
    cout << "Error : File output harus bertipe .gif" << endl;
    cout << "[RE] Masukkan path output GIF (enter untuk menonaktifkan) : ";
    getline(cin , gif_path);
    lower_input = gif_path;
    transform(lower_input.begin() , lower_input.end() , lower_input.begin() ,
::tolower);
}
```

```

cout << "-----" << endl;
cout << "Loading..." << endl;
auto start = chrono::high_resolution_clock::now();
int original_size = size_of_file(input_path);
if (target_compression > 0.0) {
    double mint = 0.0 , maxt = 10000.0;
    if (method == 1) {
        mint = 0.0;
        maxt = 10000.0;
    } else if (method == 2) {
        mint = 0.0;
        maxt = 255.0;
    } else if (method == 3) {
        mint = 0.0;
        maxt = 765.0;
    } else if (method == 4) {
        mint = 0.0;
        maxt = 24.0;
    } else {
        mint = 0.0;
        maxt = 1.0;
    }
    int best_final_size = 0;
    double tolerance_ratio = 0.0005;
    double best_threshold = threshold;
    double best_diff = std::numeric_limits<double>::max();
    for (int k = 0 ; k < 20 ; k++) {
        double mid = (mint + maxt) / 2;
        QuadNode* temp_root = BuildQuadTree(input_image , 0 , 0 , width ,
height , min_block_size , mid , method);
        vector<vector<Pixel>> temp_image(height , vector<Pixel> (width));
        for (int i = 0 ; i < height ; i++) {
            for (int j = 0 ; j < width ; j++) {
                temp_image[i][j] = input_image[i][j];
            }
        }
        ReconstructQuadTree(temp_root , temp_image);
        FreeQuadTree(temp_root);
    }
}

```

```

        string str_out = "temp_image.png";
        check = save_image(str_out , temp_image , width , height);
        int temp_size = size_of_file(str_out);
        double ratio = fabs(1.0 - ((double) temp_size / (double)
original_size));
        double diff = fabs(ratio - target_compression);
        if (diff < best_diff) {
            best_diff = diff;
            best_threshold = mid;
            best_final_size = temp_size;
        }
        if (ratio < target_compression) {
            mint = mid;
        } else {
            maxt = mid;
        }
        cout << "\r
\r" << flush;
        cout << "[COMPRESSING - " << k + 1 << "] :: " << fixed <<
setprecision(3) << ratio * 100 << "% :: Threshold " << fixed << setprecision(3)
<< best_threshold << flush;
        if (best_diff < tolerance_ratio || fabs(maxt - mint) < 0.000001 ||
k == 19) {
            cout << endl << "[Threshold Akhir] : " << fixed <<
setprecision(3) << best_threshold << endl;
            cout << "Loading..." << endl;
            break;
        }
    }
    threshold = best_threshold;
    remove("./temp_image.png");
}
QuadNode* root = BuildQuadTree(input_image , 0 , 0 , width , height ,
min_block_size , threshold , method);
vector<vector<Pixel>> res_image(height , vector<Pixel> (width));
for (int i = 0 ; i < height ; i++) {
    for (int j = 0 ; j < width ; j++) {
        res_image[i][j] = input_image[i][j];
    }
}

```

```

    }

    ReconstructQuadTree(root , res_image);
    bool saved = save_image(output_path , res_image , width , height);
    if (!saved) {
        cout << "Error : Gagal Save Gambar." << endl;
        cout << "Output Path : " << output_path << endl;
    } else {
        int node_count = 0 , max_depth = 0;
        GetQuadTreeInfo(root , node_count , max_depth , 1);
        FreeQuadTree(root);
        int final_size = size_of_file(output_path);
        double compression_ratio = (1.0 - ((double) final_size / (double)
original_size)) * 100.0;
        auto end = chrono::high_resolution_clock::now();
        int time = chrono::duration<double, std::milli>(end - start).count();
        cout << "===== SUKSES =====" << endl;
        cout << "[Waktu Eksekusi] : " << time << " ms" << endl;
        cout << "[Ukuran Gambar Awal] : " << original_size << " kb" << endl;
        cout << "[Ukuran Gambar Akhir] : " << final_size << " kb" << endl;
        cout << "[Persentase Kompresi] : " << fixed << setprecision(3) <<
compression_ratio << "%" << endl;
        cout << "[Kedalaman Tree] : " << max_depth << " depth" << endl;
        cout << "[Banyak Tree Node] : " << node_count << " node" << endl;
        cout << "[Output File Path PNG] : " << output_path << endl;
        cout << "[Output File Path GIF] : " << "N/A" << endl;
    }
    cout << "===== " << endl;
    cout << "      TERIMA KASIH TELAH MENGGUNAKAN PROGRAM INI      " << endl;
    cout << "      SAMPAI JUMPA KEMBALI!                      " << endl;
    cout << "===== " << endl;
}

```

5.2. Program quadtree.hpp

```

/* Nama      : Muhammad Raihan Nazhim Oktana & Muhammad Alfansya */
/* NIM       : 13523021 & 13523005 - Teknik Informatika           */

```

```
/* Tanggal      : Jumat, 11 April 2025          */
/* Tugas        : Stima (IF2211-24) - Tucil 2 - Divide & Conquer    */
/* File         : quadtree.hpp                   */
/* Deskripsi    : F02A - Quadtree - Header HPP      */
/* PJ F02A      : Muhammad Raihan Nazhim Oktana - 13523021      */

#ifndef QUADTREE_HPP
#define QUADTREE_HPP

#include <bits/stdc++.h>
#include "image_util.hpp"
using namespace std;

struct QuadNode {
    Pixel avg_color;
    int startx , starty;
    int blockw , blockh;
    QuadNode* child[4];
    QuadNode() {
        for (int i = 0 ; i < 4 ; i++) {
            child[i] = nullptr;
        }
    }
};

QuadNode* BuildQuadTree(vector<vector<Pixel>>& image , int startx , int starty
, int blockw , int blockh , int min_block_size , double threshold , int
method);
double ComputeBlockError(const vector<vector<Pixel>>& image , int startx , int
starty , int blockw , int blockh , int method , Pixel& average);
void ReconstructQuadTree(QuadNode* root , vector<vector<Pixel>>& res_img);
void FreeQuadTree(QuadNode* root);
void GetQuadTreeInfo(QuadNode* root , int& node_count , int& max_depth , int
cur_depth = 1);
```

```
#endif
```

5.3. Program quadtree.cpp

```
/* Nama      : Muhammad Raihan Nazhim Oktana & Muhammad Alfansya */
/* NIM       : 13523021 & 13523005 - Teknik Informatika          */
/* Tanggal   : Jumat, 11 April 2025                         */
/* Tugas     : Stima (IF2211-24) - Tucil 2 - Divide & Conquer   */
/* File      : quadtree.cpp                                     */
/* Deskripsi  : F02B - Quadtree - Implementasi CPP           */
/* PJ F02B    : Muhammad Raihan Nazhim Oktana - 13523021      */

#include <bits/stdc++.h>
#include "quadtree.hpp"
using namespace std;

static double ComputeVariance(const vector<vector<Pixel>>& image , int startx ,
int starty , int blockw , int blockh , Pixel& average) {
    long long sumr = 0 , sumg = 0 , sumb = 0;
    long long n = (long long) blockw * blockh;
    for (int i = starty ; i < starty + blockh ; i++) {
        for (int j = startx ; j < startx + blockw ; j++) {
            sumr += image[i][j].r;
            sumg += image[i][j].g;
            sumb += image[i][j].b;
        }
    }
    double meanr = (double) sumr / n;
    double meang = (double) sumg / n;
    double meanb = (double) sumb / n;
    average.r = (unsigned char) round(meanr);
    average.g = (unsigned char) round(meang);
    average.b = (unsigned char) round(meanb);
```

```

double varr = 0 , varg = 0 , varb = 0;
for (int i = starty ; i < starty + blockh ; i++) {
    for (int j = startx ; j < startx + blockw ; j++) {
        double dr = image[i][j].r - meanr;
        double dg = image[i][j].g - meang;
        double db = image[i][j].b - meanb;
        varr += dr * dr;
        varg += dg * dg;
        varb += db * db;
    }
}
varr /= n;
varg /= n;
varb /= n;
double res = (varr + varg + varb) / 3.0;
return res;
}

static double ComputeMAD(const vector<vector<Pixel>>& image , int startx , int
starty , int blockw , int blockh , Pixel& average) {
    long long sumr = 0 , sumg = 0 , sumb = 0;
    long long n = (long long) blockw * blockh;
    for (int i = starty ; i < starty + blockh ; i++) {
        for (int j = startx ; j < startx + blockw ; j++) {
            sumr += image[i][j].r;
            sumg += image[i][j].g;
            sumb += image[i][j].b;
        }
    }
    double meanr = (double) sumr / n;
    double meang = (double) sumg / n;
    double meanb = (double) sumb / n;
    average.r = (unsigned char) round(meanr);
    average.g = (unsigned char) round(meang);
}

```

```

average.b = (unsigned char) round(meanb);

double madr = 0 , madg = 0 , madb = 0;
for (int i = starty ; i < starty + blockh ; i++) {
    for (int j = startx ; j < startx + blockw ; j++) {
        madr += fabs(image[i][j].r - meanr);
        madg += fabs(image[i][j].g - meang);
        madb += fabs(image[i][j].b - meanb);
    }
}
madr /= n;
madg /= n;
madb /= n;
double res = (madr + madg + madb) / 3.0;
return res;
}

static double ComputeMaxPixelDiff(const vector<vector<Pixel>>& image , int
startx , int starty , int blockw , int blockh , Pixel& average) {
    int minr = 255 , ming = 255 , minb = 255;
    int maxr = 0 , maxg = 0 , maxb = 0;
    long long sumr = 0 , sumg = 0 , sumb = 0;
    long long n = (long long) blockw * blockh;
    for (int i = starty ; i < starty + blockh ; i++) {
        for (int j = startx ; j < startx + blockw ; j++) {
            int r = image[i][j].r;
            int g = image[i][j].g;
            int b = image[i][j].b;
            maxr = max(maxr , r);
            minr = min(minr , r);
            maxg = max(maxg , g);
            ming = min(ming , g);
            maxb = max(maxb , b);
            minb = min(minb , b);
            sumr += r;
        }
    }
    average.r = (unsigned char) round(maxr);
    average.g = (unsigned char) round(maxg);
    average.b = (unsigned char) round(maxb);
}

```

```

        sumg += g;
        sumb += b;
    }
}

double meanr = (double) sumr / n;
double meang = (double) sumg / n;
double meanb = (double) sumb / n;
average.r = (unsigned char) round(meanr);
average.g = (unsigned char) round(meang);
average.b = (unsigned char) round(meanb);
double dr = (double) (maxr - minr);
double dg = (double) (maxg - ming);
double db = (double) (maxb - minb);
double res = (dr + dg + db) / 3.0;
return res;
}

static double ComputeEntropy(const vector<vector<Pixel>>& image , int startx ,
int starty , int blockw , int blockh , Pixel& average) {
long long sumr = 0 , sumg = 0 , sumb = 0;
long long n = (long long) blockw * blockh;
for (int i = starty ; i < starty + blockh ; i++) {
    for (int j = startx ; j < startx + blockw ; j++) {
        sumr += image[i][j].r;
        sumg += image[i][j].g;
        sumb += image[i][j].b;
    }
}
double meanr = (double) sumr / n;
double meang = (double) sumg / n;
double meanb = (double) sumb / n;
average.r = (unsigned char) round(meanr);
average.g = (unsigned char) round(meang);
average.b = (unsigned char) round(meanb);
}

```

```

vector<int> vhistr(256 , 0) , vhistg(256 , 0) , vhistb(256 , 0);
for (int i = starty ; i < starty + blockh ; i++) {
    for (int j = startx ; j < startx + blockw ; j++) {
        vhistr[image[i][j].r]++;
        vhistg[image[i][j].g]++;
        vhistb[image[i][j].b]++;
    }
}
auto solve_entropy = [&](const vector<int>& hist) {
    double e = 0.0;
    for (int i = 0 ; i < 256 ; i++) {
        if (hist[i] > 0) {
            double p = (double) hist[i] / n;
            e += -p * log2(p);
        }
    }
    return e;
};
double er = solve_entropy(vhistr);
double eg = solve_entropy(vhistg);
double eb = solve_entropy(vhistb);
double res = (er + eg + eb) / 3.0;
return res;
}

double ComputeBlockError(const vector<vector<Pixel>>& image , int startx , int
starty , int blockw , int blockh , int method , Pixel& average) {
    if (method == 1) {
        return ComputeVariance(image , startx , starty , blockw , blockh ,
average);
    } else if (method == 2) {
        return ComputeMAD(image , startx , starty , blockw , blockh , average);
    } else if (method == 3) {
        return ComputeMaxPixelDiff(image , startx , starty , blockw , blockh ,
average);
}

```

```

average);

} else if (method == 4) {
    return ComputeEntropy(image, startx, starty, blockw, blockh,
average);
} else {
    return ComputeVariance(image, startx, starty, blockw, blockh,
average);
}
}

QuadNode* BuildQuadTree(vector<vector<Pixel>>& image, int startx, int starty,
, int blockw, int blockh, int min_block_size, double threshold, int method)
{
    QuadNode* node = new QuadNode();
    (*node).startx = startx;
    (*node).starty = starty;
    (*node).blockw = blockw;
    (*node).blockh = blockh;
    Pixel avg;
    double error_value = ComputeBlockError(image, startx, starty, blockw,
blockh, method, avg);
    (*node).avg_color = avg;
    if ((error_value > threshold) && (blockw > min_block_size) && (blockh >
min_block_size)) {
        int halfW = blockw / 2;
        int halfH = blockh / 2;
        (*node).child[0] = BuildQuadTree(image, startx, starty, halfW,
halfH, min_block_size, threshold, method);
        (*node).child[1] = BuildQuadTree(image, startx + halfW, starty,
blockw - halfW, halfH, min_block_size, threshold, method);
        (*node).child[2] = BuildQuadTree(image, startx, starty + halfH,
halfW, blockh - halfH, min_block_size, threshold, method);
        (*node).child[3] = BuildQuadTree(image, startx + halfW, starty +
halfH, blockw - halfW, blockh - halfH, min_block_size, threshold, method);
    }
}

```

```

    }

    return node;
}

void ReconstructQuadTree(QuadNode* root , vector<vector<Pixel>>& res_img) {
    if (root) {
        bool check_leaf = true;
        for (int i = 0 ; i < 4 ; i++) {
            if ((*root).child[i] != nullptr) {
                check_leaf = false;
                break;
            }
        }
        if (check_leaf) {
            for (int i = (*root).starty ; i < (*root).starty + (*root).blockh ;
i++) {
                for (int j = (*root).startx ; j < (*root).startx +
(*root).blockw ; j++) {
                    res_img[i][j] = (*root).avg_color;
                }
            }
        } else {
            for (int i = 0 ; i < 4 ; i++) {
                ReconstructQuadTree((*root).child[i] , res_img);
            }
        }
    }
}

void FreeQuadTree(QuadNode* root) {
    if (root) {
        for (int i = 0 ; i < 4 ; i++) {
            FreeQuadTree((*root).child[i]);
        }
    }
}

```

```

        delete root;
    }

}

void GetQuadTreeInfo(QuadNode* root , int& node_count , int& max_depth , int
cur_depth) {
    if (root) {
        node_count++;
        if (cur_depth > max_depth) {
            max_depth = cur_depth;
        }
        for (int i = 0 ; i < 4 ; i++) {
            if ((*root).child[i]) {
                GetQuadTreeInfo((*root).child[i] , node_count , max_depth ,
cur_depth + 1);
            }
        }
    }
}

```

5.4. Program image_util.hpp

```

/* Nama      : Muhammad Raihan Nazhim Oktana & Muhammad Alfansya */
/* NIM       : 13523021 & 13523005 - Teknik Informatika           */
/* Tanggal   : Jumat, 11 April 2025                         */
/* Tugas     : Stima (IF2211-24) - Tucil 2 - Divide & Conquer   */
/* File      : image_util.hpp                                */
/* Deskripsi  : F03A - Image Utility - Header HPP          */
/* PJ F03A    : Muhammad Alfansya - 13523005                */

#ifndef IMAGE_UTIL_HPP
#define IMAGE_UTIL_HPP

#include <bits/stdc++.h>

```

```

#include <sys/stat.h>
using namespace std;

struct Pixel {
    unsigned char r;
    unsigned char g;
    unsigned char b;
};

bool load_image(const string& filename , vector<vector<Pixel>>& image_data ,
int& width , int& height);
bool save_image(const string& filename , const vector<vector<Pixel>>&
image_data , int width , int height);
int size_of_file(string file_path);

#endif

```

5.5. Program image_util.cpp

```

/* Nama      : Muhammad Raihan Nazhim Oktana & Muhammad Alfansya */
/* NIM       : 13523021 & 13523005 - Teknik Informatika           */
/* Tanggal   : Jumat, 11 April 2025                         */
/* Tugas     : Stima (IF2211-24) - Tucil 2 - Divide & Conquer   */
/* File      : image_util.cpp                                */
/* Deskripsi  : F03B - Image Utility - Implementasi CPP      */
/* PJ F03B    : Muhammad Alfansya - 13523005                 */

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

#include <bits/stdc++.h>

```

```
#include "image_util.hpp"
using namespace std;

bool load_image(const string& filename , vector<vector<Pixel>>& image_data ,
int& width , int& height) {
    int channels;
    unsigned char* data = stbi_load(filename.c_str() , &width , &height ,
&channels , 3);
    if (!data) {
        return false;
    } else {
        image_data.resize(height , vector<Pixel> (width));
        size_t idx = 0;
        for (int i = 0 ; i < height ; i++) {
            for (int j = 0 ; j < width ; j++) {
                Pixel p;
                p.r = data[idx];
                p.g = data[idx + 1];
                p.b = data[idx + 2];
                image_data[i][j] = p;
                idx += 3;
            }
        }
        stbi_image_free(data);
        return true;
    }
}

bool save_image(const string& filename , const vector<vector<Pixel>>&
image_data , int width , int height) {
    vector<unsigned char> buf(width * height * 3);
    size_t idx = 0 ;
    for (int i = 0 ; i < height ; i++) {
        for (int j = 0 ; j < width ; j++) {
```

```
    buf[idx] = image_data[i][j].r;
    buf[idx + 1] = image_data[i][j].g;
    buf[idx + 2] = image_data[i][j].b;
    idx += 3;
}
}

int stride_in_bytes = width * 3;
if (!stbi_write_png(filename.c_str(), width, height, 3, buf.data(), stride_in_bytes)) {
    return false;
} else {
    return true;
}
}

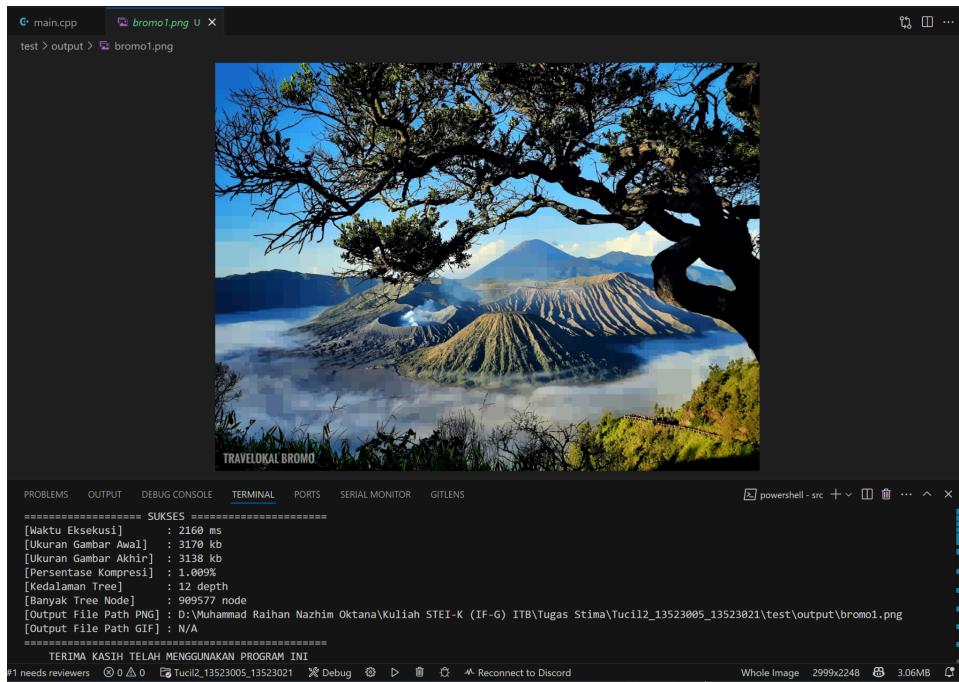
int size_of_file(string file_path) {
    struct stat file_info;
    int res = stat(file_path.c_str(), &file_info);
    if (res == 0) {
        long file_size = file_info.st_size;
        return ceil((double) file_size/1024);
    } else {
        return -1;
    }
}
```

BAB VI

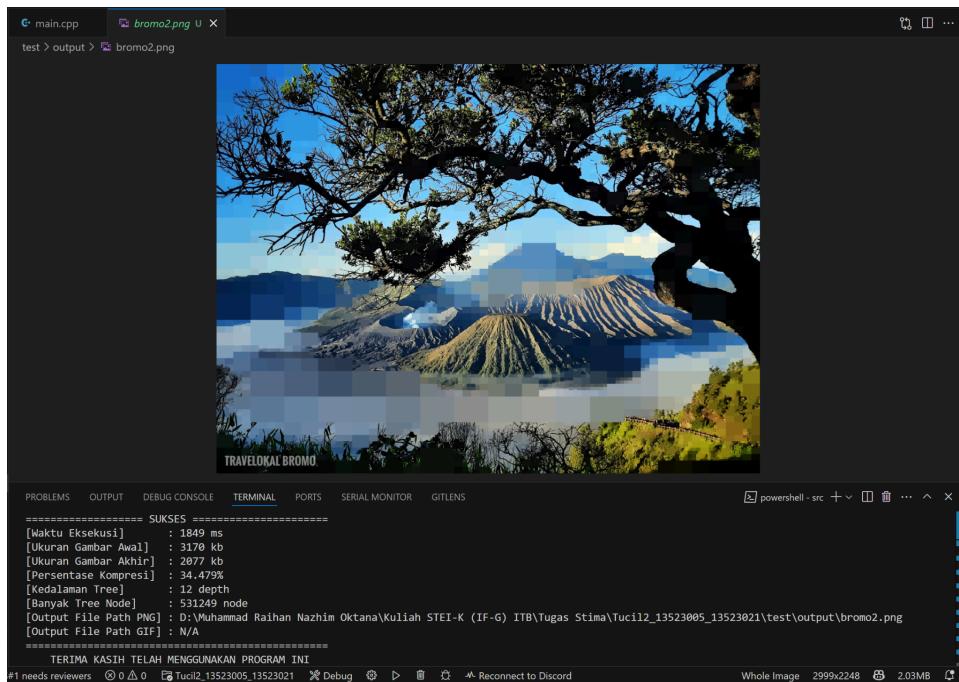
TEST CASE

6.1. Kumpulan Test Case Hasil Gambar Optimal

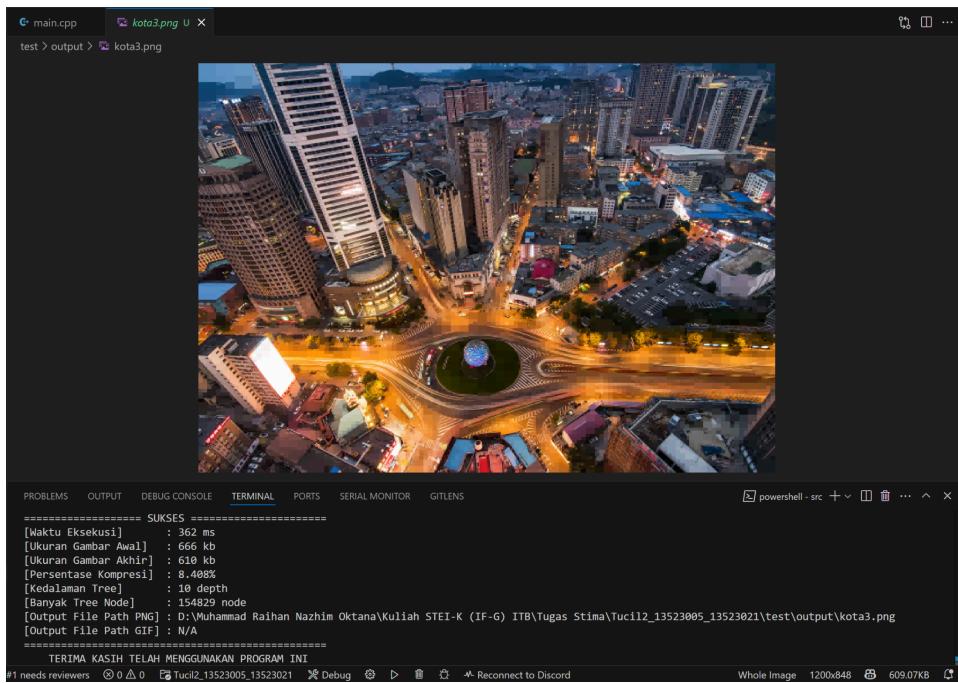
6.1.1. Menggunakan Metode 1



6.1.2. Menggunakan Metode 2



6.1.3. Menggunakan Metode 3



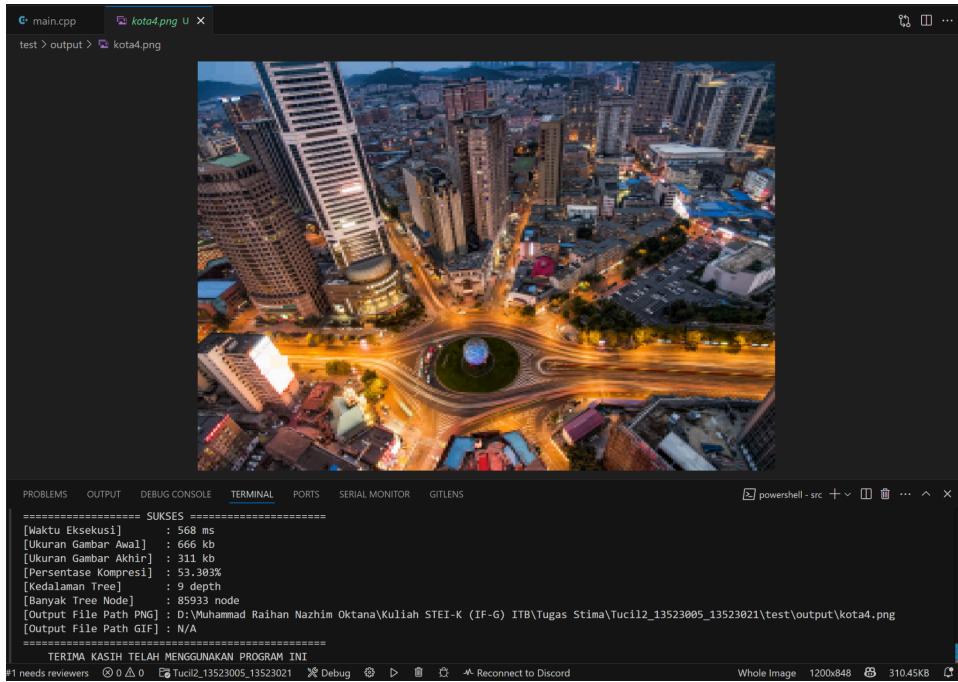
```

main.cpp  kota3.png  x
test > output > kota3.png

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SERIAL MONITOR  GITLENS
===== SUKSES =====
[Waktu Esekusi] : 362 ms
[Ukuran Gambar Awal] : 666 kb
[Ukuran Gambar Akhir] : 610 kb
[Percentase Kompressi] : 8.408%
[Kedalaman Tree] : 10 depth
[Banyak Tree Node] : 154829 node
[Output File Path PNG] : D:\Muhammad Raihan Nazhim Oktana\Kuliah STEI-K (IP-G) ITB\Tugas Stima\Tucil2_13523005_13523021\test\output\kota3.png
[Output File Path GIF] : N/A
===== TERIMA KASIH TELAH MENGGUNAKAN PROGRAMINI
#1 needs reviewers 0 0 0 Tucil2_13523005_13523021  Debug  Reconnect to Discord
Whole Image 1200x848 609.07KB

```

6.1.4. Menggunakan Metode 4



```

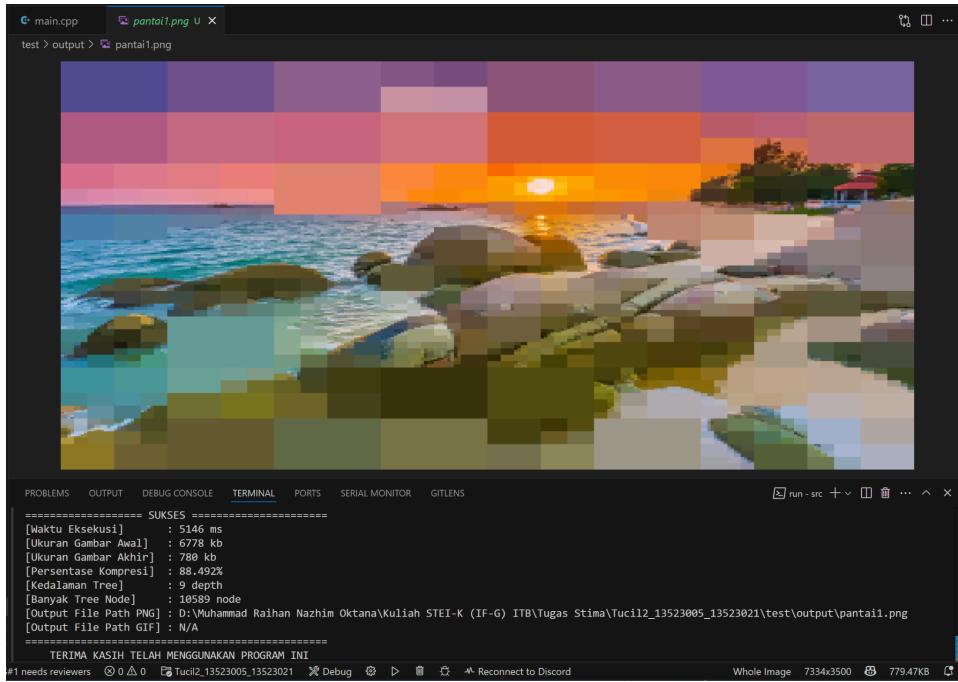
main.cpp  kota4.png  x
test > output > kota4.png

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SERIAL MONITOR  GITLENS
===== SUKSES =====
[Waktu Esekusi] : 568 ms
[Ukuran Gambar Awal] : 666 kb
[Ukuran Gambar Akhir] : 311 kb
[Percentase Kompressi] : 53.303%
[Kedalaman Tree] : 9 depth
[Banyak Tree Node] : 85933 node
[Output File Path PNG] : D:\Muhammad Raihan Nazhim Oktana\Kuliah STEI-K (IP-G) ITB\Tugas Stima\Tucil2_13523005_13523021\test\output\kota4.png
[Output File Path GIF] : N/A
===== TERIMA KASIH TELAH MENGGUNAKAN PROGRAMINI
#1 needs reviewers 0 0 0 Tucil2_13523005_13523021  Debug  Reconnect to Discord
Whole Image 1200x848 310.45KB

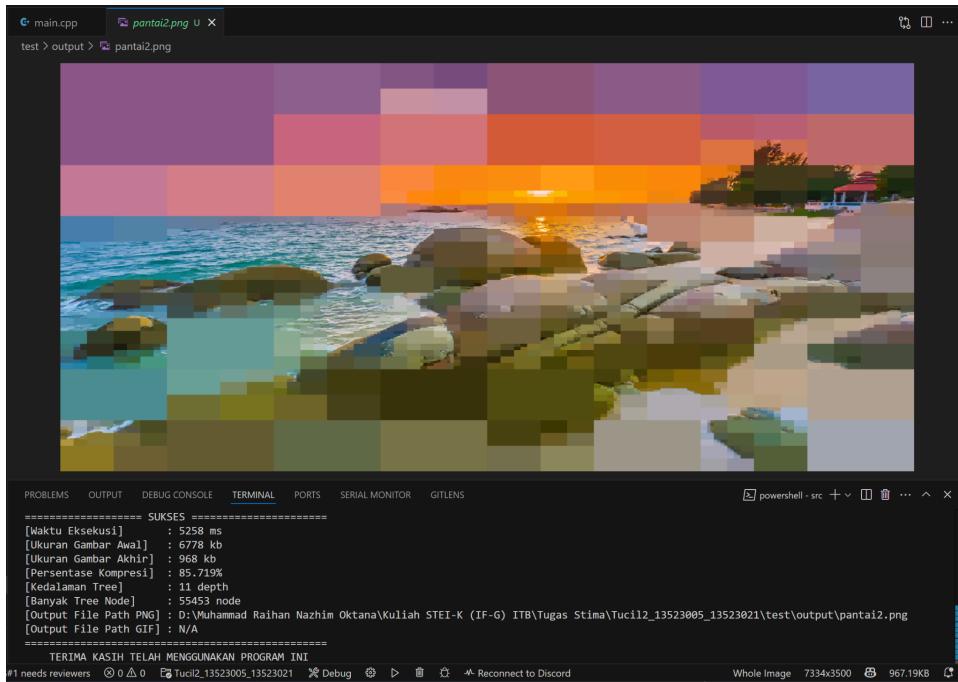
```

6.2. Kumpulan Test Case Hasil Kompresi Optimal

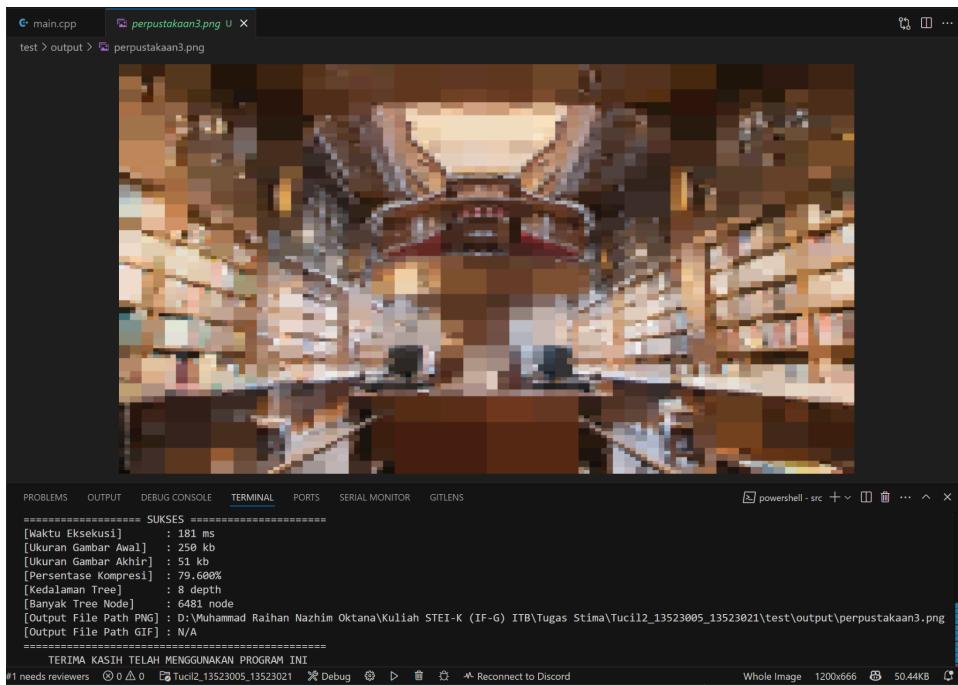
6.2.1. Menggunakan Metode 1



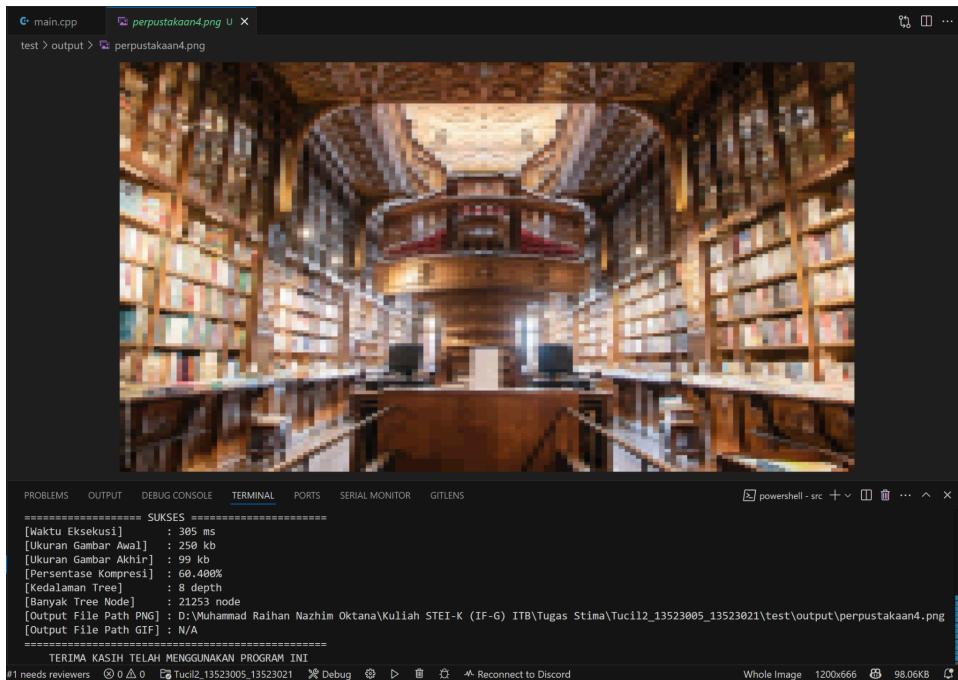
6.2.2. Menggunakan Metode 2



6.2.3. Menggunakan Metode 3

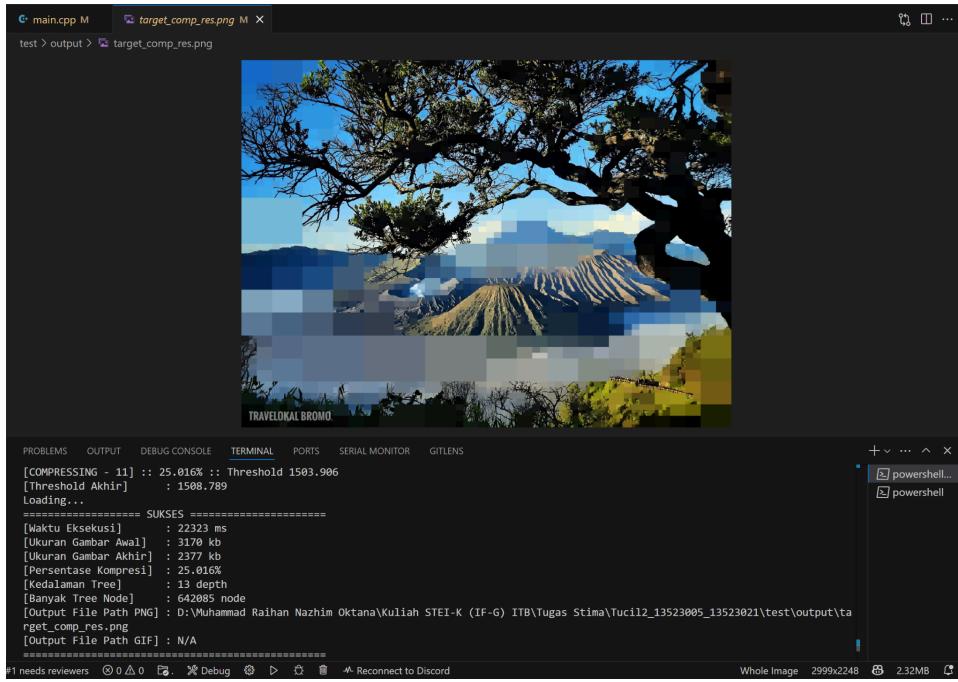


6.2.4. Menggunakan Metode 4

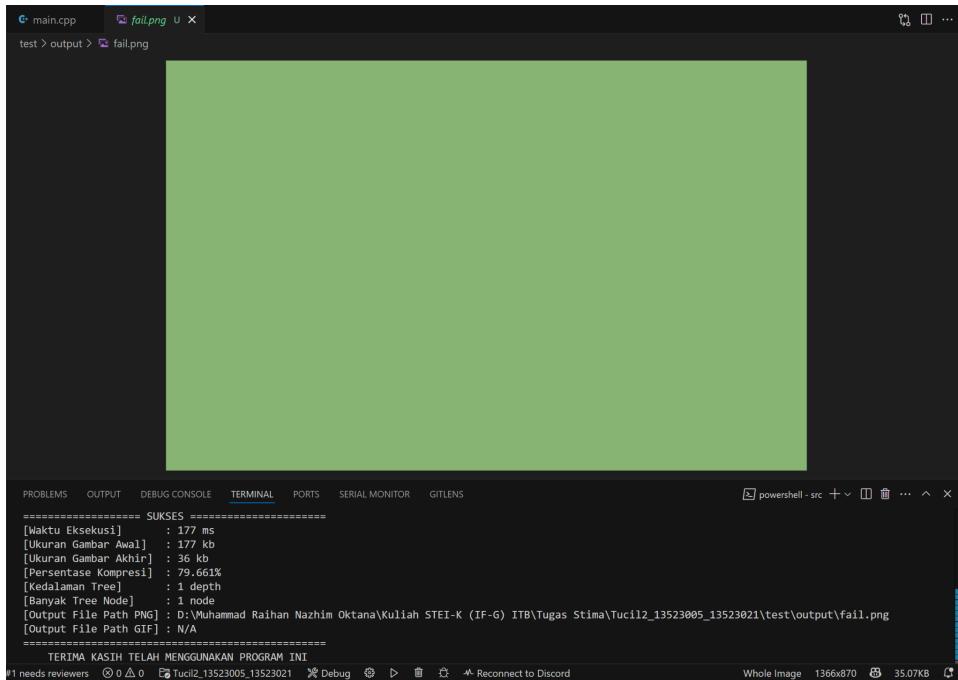


6.3. Kumpulan Test Case Lainnya

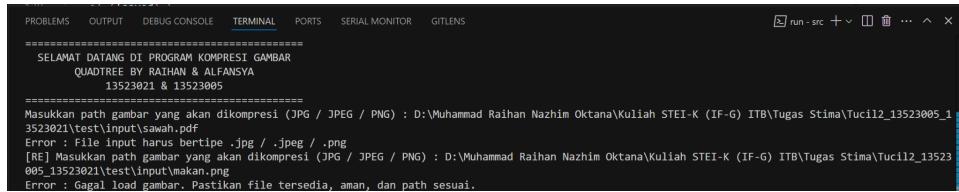
6.3.1. Hasil Gambar Bonus Target Compression



6.3.2. Hasil Gambar Hanya 1 Potongan



6.3.3. Error Handling 1 : Format File Tidak Tepat / File Tidak Tersedia



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR GITLENS
===== SELAMAT DATANG DI PROGRAM KOMPRESI GAMBAR =====
QUADTREE BY RAIHAN & ALFANSYA
13523021 & 13523005
=====
Masukkan path gambar yang akan dikompresi (JPG / JPEG / PNG) : D:\Muhammad Raihan Nazhim Oktana\Kuliah STEI-K (IF-G) ITB\Tugas Stima\Tucil2_13523005_1
3523021\test\input\sawah.pdf
Error : File input harus bertipe .jpg / .jpeg / .png
[RE] Masukkan path gambar yang akan dikompresi (JPG / JPEG / PNG) : D:\Muhammad Raihan Nazhim Oktana\Kuliah STEI-K (IF-G) ITB\Tugas Stima\Tucil2_13523005_1
005_13523021\test\input\makanan.png
Error : Gagal load gambar. Pastikan file tersedia, aman, dan path sesuai.

```

6.3.4. Error Handling 2 : Pilihan Metode Tidak Valid



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR GITLENS
=====
Pilihan Metode Perhitungan Error :
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
Masukkan metode yang ingin digunakan (angka) : 6
Error : Metode tidak valid. Hanya boleh 1 - 4 (SSIM belum tersedia).
[RE] Masukkan metode yang ingin digunakan (angka) : 5
Error : Metode tidak valid. Hanya boleh 1 - 4 (SSIM belum tersedia).
[RE] Masukkan metode yang ingin digunakan (angka) : -1
Error : Metode tidak valid. Hanya boleh 1 - 4 (SSIM belum tersedia).
[RE] Masukkan metode yang ingin digunakan (angka) :

```

6.3.5. Error Handling 3 : Nilai Threshold Tidak Valid

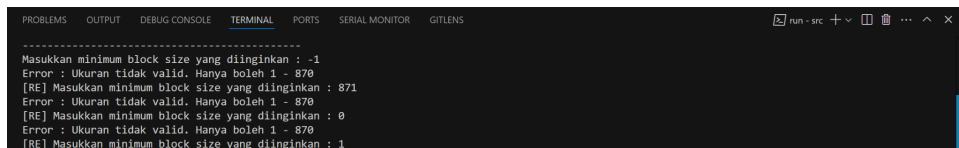


```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR GITLENS
=====
Saran Range Threshold :
Range Threshold Metode MAD = 0 <= ... <= 255
=====
Masukkan nilai threshold (ambang batas) : -1
Error : Threshold tidak valid. Threshold tidak boleh negatif, usahakan sesuai saran.
[RE] Masukkan nilai threshold (ambang batas) : -10
Error : Threshold tidak valid. Threshold tidak boleh negatif, usahakan sesuai saran.
[RE] Masukkan nilai threshold (ambang batas) : 256
Masukkan minimum block size yang diinginkan :

```

6.3.6. Error Handling 4 : Minimum Block Size Tidak Valid

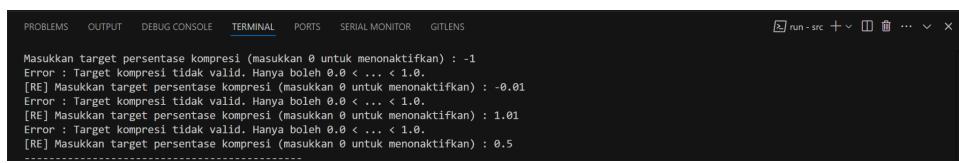


```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR GITLENS
=====
Masukkan minimum block size yang diinginkan : -1
Error : Ukuran tidak valid. Hanya boleh 1 - 870
[RE] Masukkan minimum block size yang diinginkan : 871
Error : Ukuran tidak valid. Hanya boleh 1 - 870
[RE] Masukkan minimum block size yang diinginkan : 0
Error : Ukuran tidak valid. Hanya boleh 1 - 870
[RE] Masukkan minimum block size yang diinginkan : 1

```

6.3.7. Error Handling 5 : Target Compression Tidak Valid



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR GITLENS
=====
Masukkan target persentase kompresi (masukkan 0 untuk menonaktifkan) : -1
Error : Target kompresi tidak valid. Hanya boleh 0.0 < ... < 1.0.
[RE] Masukkan target persentase kompresi (masukkan 0 untuk menonaktifkan) : -0.01
Error : Target kompresi tidak valid. Hanya boleh 0.0 < ... < 1.0.
[RE] Masukkan target persentase kompresi (masukkan 0 untuk menonaktifkan) : 1.01
Error : Target kompresi tidak valid. Hanya boleh 0.0 < ... < 1.0.
[RE] Masukkan target persentase kompresi (masukkan 0 untuk menonaktifkan) : 0.5

```

LAMPIRAN

1. Pranala Github : https://github.com/RNXFreeze/Tucil2_13523005_13523021
2. Tabel Checklist :

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.	✓	