Laporan Tugas Kecil 2

Disusun untuk memenuhi tugas besar mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2024/2025

Oleh

Stefan Mattew Susanto 13523020

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

Daftar Isi

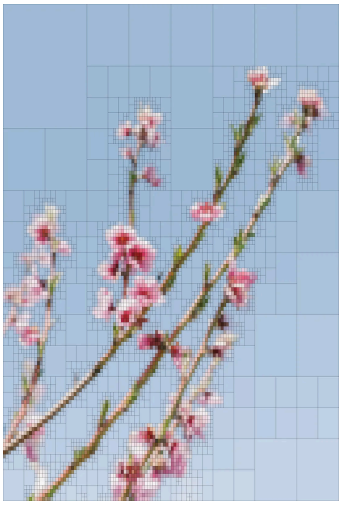
Deskripsi Masalah

1.1. Algoritma Divide and conquer

Divide membagi membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan awal namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama),Conquer: menyelesaikan (solve) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar). Algoritma Divide and Conquer menggabungkan solusi masing masing upa-persoalan sehingga membentuk solusi persoalan awal.

Objek pesoalan yang dibagi ialah input ataupun instance suatu perosalan, misalnya tabel, matriks, gambar, ataupun lainnya. Setiap upa persoalan memiliki karakteristik yang sama namun dengan skala/ukuran yang lebih kecil. Dalam algoritma divide and conquer lebih terfokus pada logika pengulangan atau rekursif.

1.2. Quadttree pada kompresi gambar



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

Penyelesaian

Langkah penyelesaian dengan Divide and conquer:

1. Program akan meminta input dari user berupa alamat absolut gambar, metode perhitungan error, ambang batas, ukuran blok minimum, target ukuran kompresi, alamat absolut gambar hasil kompresi.
2. Program akan memulai dengan membaca image dari alamat absolut dengan menggunakan stbi_load. Jika image berhasil dibaca maka program akan dilanjutkan namun jika tidak maka program akan berhenti. Program akan memulai waktu perhitungan.Program akan melakukan quadtree

DIVIDE

3. Gambar memiliki ukuran m x n. Blok akan dibagi menjadi 4 sub blok (upa-persoalan) yang memiliki panjang atau lebar yang sama jika panjang atau lebar genap, jika ganjil maka bagian sebelah kanan atau bawah akan +1 dari kiri atau atas.
   Sub blok akan memiliki karakteristik yang sama dengan permaslahan awal, namun dengan ukuran yang lebih kecil.

DIVIDE AND CONQUER

4. Tiap sub blok yang telah dibagi akan dihitung nilai error dengan salah satu metode perhitungan.
5. Jika nilai error lebih besar atau sama dengan threshold maka blok tersebut akan membagi blok menjadi 4 subblok secara rekursif. Ketika blok dibagi menjadi 4 maka kedalaman pohon akan bertambah 1 dan simpul akan bertambah.
6. Blok tersebut dibagi selama saat ukuran blok masih lebih dari minSize.
7. Blok akan berhenti membagi sub blok ketika nilai error lebih kecil dari nilai threshold atau ketika ukuran blok telah mencapai minSize.
8. Blok yang berhenti membagi akan masuk tahap conquer/penyelesaian.

Conquer

9. Setelah blok telah selesai dibagi sesuai dengan minsize dan threshold, blok akan melakukan normalisasi warna dengan mengubah blok tersebut menjadi satu warna berdasarkan rata rata warna tiap piksel pada blok tersebut. Hal ini dilakukan dengan fungsi normColor.

Combine

10. Setiap blok yang telah berhasil diselesaikan dengan divide and conquer akan digabungkan menjadi satu solusi. Tiap sub blok akan disusun sesuai dengan koordinat x dan y.
11. Hasil akhir dari proses ini akan menghasilkan gambar yang telah dikompresi dengan menggunakan quadtree. Jenis gambar yang dihasilkan akan sesuai dengan extension output yang dipilih berdasarkan path.

12. Blok akan melanjutkan berbagai output ,seperti output waktu yang digunakan, ukuran gambar sebelum dan sesudah, kedalaman pohon dan banyak simpul

Implementasi Program

3.1. Spesifikasi Teknis Program

3.1.1.Struktur data

```
└── Tucil2_13523020
        └── src
            ├──inputOutput.h
            ├──inputOutput.c
            ├──quadtreeCompression.h
            ├──quadtreeCompression.c
            ├──errMethod.h
            ├──errMethod.c
            ├──stb_image.h
            ├──stb_image_write.h
            └── main.c

        └── doc
            └── Tucil2_13523020_Stefan Mattew Susanto,pdf
        └── bin
            └── main.exe
        └── test
            ├── test1.jpg
            ├── test2.jpg
            ├── test3.jpg
            ├── test4.jpg
            ├── test5.jpg
            ├── test6.jpg
            ├── test7.jpg
            ├── test8.jpg
            ├── test9.jpg
            ├── test10.jpg
            ├── test11.jpeg
            ├── test12.jpeg
            └── test13.png
        └── README.md
```

3.1.2. Fungsi dan prosedur

Fungsi/ Prosedur
`double errMeasure(unsigned char* image, int x, int y, int blockWidth, int blockHeight, int width, int channels, int method)`
`double computeVariance(unsigned char* image, int x, int y, int blockWidth, int blockHeight, int width, int channels)`
`double computeMAD(unsigned char* image, int x, int y, int blockWidth, int blockHeight, int width, int channels)`
`double computeMPD(unsigned char* image, int x, int y, int blockWidth, int blockHeight, int width, int channels)`
`double computeEntropy(unsigned char* image, int x, int y, int blockWidth, int blockHeight, int width, int channels)`
`void normColor(unsigned char* compressed, int x, int y, int blockWidth, int blockHeight, int width, int channels, unsigned char* image)`
`void quadtreeCompression(unsigned char* image, unsigned char* compressed, int x, int y, int blockWidth, int blockHeight, int width, int channels, double threshold, int minSize, int depth,int method)`
`long getFileSize(const char* filename)`
`void toLowerCase(char* str)`
`bool cekInputFormat(const char* filename)`

```
bool cekOutputFormat(const char* str, const char* suffix)
int main()
```

3.1.3. Source Code Program

Pseudocode errMethod.c

```
function errMeasure(image, x, y, blockWidth, blockHeight, width, channels, method) -> double
    if method = 1 then
        return computeVariance(image, x, y, blockWidth, blockHeight, width, channels)
    else if method = 2 then
        return computeMAD(image, x, y, blockWidth, blockHeight, width, channels)
    else if method = 3 then
        return computeMPD(image, x, y, blockWidth, blockHeight, width, channels)
    else if method = 4 then
        return computeEntropy(image, x, y, blockWidth, blockHeight, width, channels)
    else
        return computeVariance(image, x, y, blockWidth, blockHeight, width, channels)


function computeVariance(image, x, y, blockWidth, blockHeight, width, channels) -> double
    declare mean[3] <- {0, 0, 0}
    declare banyakPiksel <- blockWidth * blockHeight

    for i from 0 to blockHeight - 1 do
        for j from 0 to blockWidth - 1 do
            index <- ((y + i) * width + (x + j)) * channels
            for c from 0 to channels - 1 do
                mean[c] <- mean[c] + image[index + c]

    for c from 0 to 2 do
        mean[c] <- mean[c] / banyakPiksel

    declare variance[3] <- {0, 0, 0}
    for i from 0 to blockHeight - 1 do
        for j from 0 to blockWidth - 1 do
            index <- ((y + i) * width + (x + j)) * channels
            for c from 0 to channels - 1 do
                variance[c] <- variance[c] + (image[index + c] - mean[c])^2
```

```
    hasil <- (variance[0] + variance[1] + variance[2]) / (3 * banyakPiksel)
    return hasil

function computeMAD(image, x, y, blockWidth, blockHeight, width, channels) -> double
    declare mean[3] <- {0, 0, 0}
    declare banyakPiksel <- blockWidth * blockHeight

    for i from 0 to blockHeight - 1 do
        for j from 0 to blockWidth - 1 do
            index <- ((y + i) * width + (x + j)) * channels
            for c from 0 to channels - 1 do
                mean[c] <- mean[c] + image[index + c]

    for c from 0 to 2 do
        mean[c] <- mean[c] / banyakPiksel

    declare mad[3] <- {0, 0, 0}
    for i from 0 to blockHeight - 1 do
        for j from 0 to blockWidth - 1 do
            index <- ((y + i) * width + (x + j)) * channels
            for c from 0 to channels - 1 do
                mad[c] <- mad[c] + abs(image[index + c] - mean[c])

    hasil <- (mad[0] + mad[1] + mad[2]) / (3 * banyakPiksel)
    return hasil

function computeMPD(image, x, y, blockWidth, blockHeight, width, channels) -> double
    declare minVal[3] <- {255, 255, 255}
    declare maxVal[3] <- {0, 0, 0}

    for i from 0 to blockHeight - 1 do
        for j from 0 to blockWidth - 1 do
            index <- ((y + i) * width + (x + j)) * channels
            for c from 0 to 2 do
                val <- image[index + c]
                if val < minVal[c] then
                    minVal[c] <- val
                if val > maxVal[c] then
                    maxVal[c] <- val

    diffR <- maxVal[0] - minVal[0]
    diffG <- maxVal[1] - minVal[1]
```

```
    diffB <- maxVal[2] - minVal[2]

    hasil <- (diffR + diffG + diffB) / 3.0
    return hasil

function computeEntropy(image, x, y, blockWidth, blockHeight, width, channels) -> double
    declare histogram[3][256] <- all 0
    declare banyakPiksel <- blockWidth * blockHeight

    for i from 0 to blockHeight - 1 do
        for j from 0 to blockWidth - 1 do
            index <- ((y + i) * width + (x + j)) * channels
            for c from 0 to 2 do
                val <- image[index + c]
                histogram[c][val] <- histogram[c][val] + 1

    declare entropy[3] <- {0.0, 0.0, 0.0}
    for c from 0 to 2 do
        for i from 0 to 255 do
            if histogram[c][i] > 0 then
                p <- histogram[c][i] / banyakPiksel
                entropy[c] <- entropy[c] - p * log2(p)

    return (entropy[0] + entropy[1] + entropy[2]) / 3.0
```

Pseudocode quadtreeCompression.c

```
procedure normColor(input image, input x, input y, input blockWidth, input blockHeight, input
width, input channels, output compressed)
    deklarasi sum[3] <- {0, 0, 0}
    banyakPiksel <- blockWidth * blockHeight

    for i from 0 to blockHeight - 1 do
        for j from 0 to blockWidth - 1 do
            index <- ((y + i) * width + (x + j)) * channels
            for c from 0 to channels - 1 do
                sum[c] <- sum[c] + image[index + c]

    for c from 0 to channels - 1 do
        sum[c] <- sum[c] / banyakPiksel
```

```
    for i from 0 to blockHeight - 1 do
        for j from 0 to blockWidth - 1 do
            index <- ((y + i) * width + (x + j)) * channels
            for c from 0 to channels - 1 do
                compressed[index + c] <- sum[c]


procedure quadtreeCompression(input image, output compressed, input x, input y, input
blockWidth, input blockHeight, input width, input channels, input threshold, input minSize,
input depth, input method)
    if depth > maxDepth then
        maxDepth <- depth

    nodeCount <- nodeCount + 1

    if blockWidth <= minSize or blockHeight <= minSize then
        call normColor(image, x, y, blockWidth, blockHeight, width, channels, compressed)
        return

    err <- errMeasure(image, x, y, blockWidth, blockHeight, width, channels, method)

    if err < threshold then
        call normColor(image, x, y, blockWidth, blockHeight, width, channels, compressed)
        return

    halfWidth <- blockWidth / 2
    halfHeight <- blockHeight / 2

    call quadtreeCompression(image, compressed, x, y, halfWidth, halfHeight, width, channels,
threshold, minSize, depth + 1, method)
    call quadtreeCompression(image, compressed, x + halfWidth, y, blockWidth - halfWidth,
halfHeight, width, channels, threshold, minSize, depth + 1, method)
    call quadtreeCompression(image, compressed, x, y + halfHeight, halfWidth, blockHeight -
halfHeight, width, channels, threshold, minSize, depth + 1, method)
    call quadtreeCompression(image, compressed, x + halfWidth, y + halfHeight, blockWidth -
halfWidth, blockHeight - halfHeight, width, channels, threshold, minSize, depth + 1, method)
```

Pseudocode inputOutput.c

```
function getFileSize(filename) -> long
    statStruct <- stat(filename)
    if (statStruct exists) then
        return statStruct.size
```

```
    else
        return -1
    end if
end function


procedure toLowerCase(str)
    for each character in str do
        character <- tolower(character)
    end for
end procedure

function cekInputFormat(filename) -> boolean
    ext <- get extension from filename
    if (ext is null) then
        return false
    end if

    lowerExt <- copy ext and convert to lowercase

    return (lowerExt = ".jpg") OR (lowerExt = ".jpeg") OR (lowerExt = ".png")
end function

function cekOutputFormat(str, suffix) -> boolean
    if (str is null OR suffix is null) then
        return false
    end if

    if (length(suffix) > length(str)) then
        return false
    end if

    return (substring of str at the end matches suffix)
end function
```

Procedure main.c

```
procedure Main()
    declare imagePath as string[256]
    declare outputPath as string[256]
    declare cekValidInput as boolean

    print "Masukkan path gambar (.jpg/.jpeg/.png):"
    input imagePath
    remove newline from imagePath
```

```
if (NOT cekInputFormat(imagePath)) then
    print "Ekstensi file input tidak valid. Harus .jpg, .jpeg, atau .png"
    return
end if

print "Metode yang dapat digunakan:"
print "1. Variance"
print "2. Mean Absolute Deviation"
print "3. Max Pixel Difference"
print "4. Entropy"
print "Pilih metode yang ingin digunakan:"
input method

print "Masukkan nilai threshold variansi:"
input threshold

print "Masukkan ukuran minimum blok:"
input minSize

print "Masukkan target kompresi:"
input targetCompress

print "Masukkan path gambar hasil kompresi:"
input outputPath
remove newline from outputPath

if (NOT cekInputFormat(outputPath)) then
    print "Ekstensi file output tidak valid. Harus .jpg, .jpeg, atau .png"
    return
end if

startTime <- current clock time

load image from imagePath into image, width, height, channels
if (image is null) then
    print "Gagal membaca gambar!"
    return
end if

allocate compressed with size width * height * channels
copy image to compressed

oriSize <- getFileSize(imagePath)

if (targetCompress > 0.0) then
    thresholdNow <- 0.0
    step <- 50.0
    tolerance <- 0.005
    maxAttempt <- 20
```

```
    attempt <- 0
    allocate temp with size width * height * channels
    declare tempPath as string[20]

    while (attempt < maxAttempt)
        mid <- (low + high) / 2.0
        copy image to temp
        nodeCount <- 0
        maxDepth <- 0

        call quadtreeCompression(image, temp, 0, 0, width, height, width, channels, mid,
minSize, 1, method)

        if (output is JPG or JPEG) then
            tempPath <- "temp.jpg"
            write JPG image tempPath from temp
        else
            tempPath <- "temp.png"
            write PNG image tempPath from temp
        end if

        tempSize <- getFileSize(tempPath)
        achieved <- 1.0 - (tempSize / oriSize)

        if (abs(achieved - targetCompress) < tolerance) then
            break
        end if

        if (thresholdNow == 0.0) then
            thresholdNow <- step
        else if (achieved < targetCompress) then
            thresholdNow <- thresholdNow + step
        else
            break
        end if

        attempt <- attempt + 1
    end while

    step <- step / 2.0

    while (attempt < maxAttempt)
        copy image to temp
        nodeCount <- 0
        maxDepth <- 0

        call quadtreeCompression(image, temp, 0, 0, width, height, width, channels,
thresholdNow, minSize, 1, method)

        write temp image as per output format to tempPath
```

```
            tempSize <- getFileSize(tempPath)
            achieved <- 1.0 - (tempSize / oriSize)

            if (abs(achieved - targetCompress) < tolerance) then
                break
            end if

            if (achieved < targetCompress) then
                thresholdNow <- thresholdNow + step
            else
                thresholdNow <- max(0.0, thresholdNow - step)
            end if

            step <- step / 2.0
            attempt <- attempt + 1
        end while

        threshold <- thresholdNow
        free temp
        delete file tempPath
    end if

    nodeCount <- 0
    maxDepth <- 0
    call quadtreeCompression(image, compressed, 0, 0, width, height, width, channels,
threshold, minSize, 1, method)

    if (output is JPG or JPEG) then
        write JPG image to outputPath from compressed
    else if (output is PNG) then
        write PNG image to outputPath from compressed
    else
        print "Format output tidak dikenali"
        return
    end if

    print "Gambar berhasil dikompres ke:", outputPath

    endTime <- current clock time
    executionTime <- (endTime - startTime) / CLOCKS_PER_SECOND
    print "Waktu eksekusi:", executionTime, "second"

    compressSize <- getFileSize(outputPath)
    if (oriSize > 0 AND compressSize > 0) then
        compressionRatio <- 1.0 - (compressSize / oriSize)
        print "Ukuran gambar sebelum:", oriSize, "byte"
        print "Ukuran gambar setelah:", compressSize, "byte"
        print "Persentase kompresi:", compressionRatio * 100, "%"
    end if
```

```
    print "Kedalaman pohon:", maxDepth
    print "Banyak simpul:", nodeCount

    free image
    free compressed
end procedure
```

Analisis dan Pengujian

4.1. Kasus Uji:

1. Percobaan dilakukan dengan metode = variance threshold= 200 dengan minsize = 8 tanpa menggunakan target compress.

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 1
Masukkan nilai threshold variansi: 200
Masukkan ukuran minimum blok: 8
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test1.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test1.jpg')
Waktu eksekusi: 47.553 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 1533498 byte
Persentase Kompresi: 58.76%%
Kedalaman pohon: 10
Banyak simpul: 210209
```

2. Percobaan dilakukan dengan metode = variance threshold= 50 dengan minsize = 8 tanpa menggunakan target compress.

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 1
Masukkan nilai threshold variansi: 50
Masukkan ukuran minimum blok: 8
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test2.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test2.jpg')
Waktu eksekusi: 53.356 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 1679243 byte
Persentase Kompresi: 54.84%%
Kedalaman pohon: 10
Banyak simpul: 298637
```

3. Percobaan dilakukan dengan metode = variance threshold= 50 dengan minsize = 4 tanpa menggunakan target compress.

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 1
Masukkan nilai threshold variansi: 50
Masukkan ukuran minimum blok: 4
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test3.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test3.jpg')
Waktu eksekusi: 59.926 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 2208138 byte
Persentase Kompresi: 40.62%%
Kedalaman pohon: 11
Banyak simpul: 1028709
```

4. Percobaan dilakukan dengan metode = Mean Absolute Deviation (MAD) threshold= 10 dengan minsize = 8 tanpa menggunakan target compress.

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 2
Masukkan nilai threshold variansi: 10
Masukkan ukuran minimum blok: 8
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test4.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test4.jpg')
Waktu eksekusi: 9.404 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 1559813 byte
Persentase Kompresi: 58.05%%
Kedalaman pohon: 10
Banyak simpul: 223197
```

5. Percobaan dilakukan dengan metode = Mean Absolute Deviation (MAD) threshold= 10 dengan minsize = 4 tanpa menggunakan target compress.

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 2
Masukkan nilai threshold variansi: 10
Masukkan ukuran minimum blok: 4
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test5.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test5.jpg')
Waktu eksekusi: 10.161 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 1987420 byte
Persentase Kompresi: 46.55%%
Kedalaman pohon: 11
Banyak simpul: 673101
```

6. Percobaan dilakukan dengan metode =Max Pixel Difference threshold= 10 dengan minsize = 8 tanpa menggunakan target compress.

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 3
Masukkan nilai threshold variansi: 10
Masukkan ukuran minimum blok: 8
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test6.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test6.jpg')
Waktu eksekusi: 6.938 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 1704630 byte
Persentase Kompresi: 54.16%%
Kedalaman pohon: 10
Banyak simpul: 349333
```

7. Percobaan dilakukan dengan metode =Max Pixel Difference threshold= 5 dengan minsize = 4 tanpa menggunakan target compress.

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 3
Masukkan nilai threshold variansi: 5
Masukkan ukuran minimum blok: 4
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test7.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test7.jpg')
Waktu eksekusi: 7.818 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 2285344 byte
Persentase Kompresi: 38.54%%
Kedalaman pohon: 11
Banyak simpul: 1397781
```

8. Percobaan dilakukan dengan metode = entropy threshold= 5 dengan minsize = 4 tanpa menggunakan target compress.

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 4
Masukkan nilai threshold variansi: 5
Masukkan ukuran minimum blok: 4
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test8.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test8.jpg')
Waktu eksekusi: 11.354 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 2085902 byte
Persentase Kompresi: 43.91%%
Kedalaman pohon: 11
Banyak simpul: 735273
```

9. Percobaan dilakukan dengan metode =variance threshold= 12345678 (random karena sebenarnya tidak dipakai) dengan minsize = 4 dengan menggunakan target compress = 0.5484 (54.84%) .

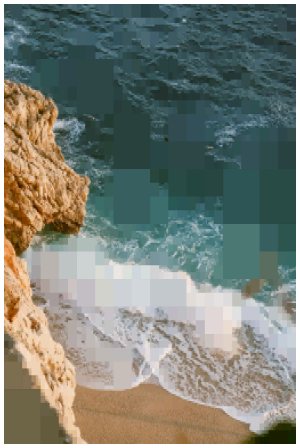
```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 1
Masukkan nilai threshold variansi: 123456789
Masukkan ukuran minimum blok: 8
Masukkan target kompresi: 0.5484
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test9.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test9.jpg')
Waktu eksekusi: 196.935 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 1679243 byte
Persentase Kompresi: 54.84%%
Kedalaman pohon: 10
Banyak simpul: 298637
```

10. Percobaan dilakukan dengan metode = variance threshold= 250 dengan minsize = 16 tanpa menggunakan target compress.

```
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path1.jpg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 1
Masukkan nilai threshold variansi: 500
Masukkan ukuran minimum blok: 16
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test10.jpg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test10.jpg')
Waktu eksekusi: 44.209 second
Ukuran gambar sebelum: 3718573 byte
Ukuran gambar setelah: 911176 byte
Persentase Kompresi: 75.50%%
Kedalaman pohon: 9
Banyak simpul: 37493
```



11. Percobaan menggunakan gambar jpeg

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/FotoDiri.jpeg
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 1
Masukkan nilai threshold variansi: 10
Masukkan ukuran minimum blok: 4
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test11.jpeg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test11.jpeg')
Waktu eksekusi: 3.304 second
Ukuran gambar sebelum: 185538 byte
Ukuran gambar setelah: 117988 byte
Persentase Kompresi: 36.41%%
Kedalaman pohon: 10
Banyak simpul: 83385
```

12. Percobaan gambar menggunakan png dan output berupa jpeg

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path2.png
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 1
Masukkan nilai threshold variansi: 10
Masukkan ukuran minimum blok: 4
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test12.jpeg

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test12.jpeg')
Waktu eksekusi: 49.009 second
Ukuran gambar sebelum: 11403122 byte
Ukuran gambar setelah: 935383 byte
Persentase Kompresi: 91.80%%
Kedalaman pohon: 12
Banyak simpul: 349497
```

13. Percobaan gambar menggunakan png dan output berupa png.

```
PS C:\coding\Tingkat 2\Tucil STIMA 2> ./quadtree_compression.exe
Masukkan path gambar (.jpg/.jpeg/.png): C:/coding/Tingkat 2/Tucil STIMA 2/path2.png
Metode yang dapat digunakan:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Pilih metode yang ingin digunakan: 1
Masukkan nilai threshold variansi: 10
Masukkan ukuran minimum blok: 4
Masukkan target kompresi: 0
Masukkan path gambar hasil kompresi: C:/coding/Tingkat 2/Tucil STIMA 2/test13.png

Gambar berhasil dikompres ('C:/coding/Tingkat 2/Tucil STIMA 2/test13.png')
Waktu eksekusi: 48.968 second
Ukuran gambar sebelum: 11403122 byte
Ukuran gambar setelah: 2136502 byte
Persentase Kompresi: 81.26%%
Kedalaman pohon: 12
Banyak simpul: 349497
```

4.2. Analisis:

1. Kompleksitas waktu dalam algoritma Quadtree
    - Ukuran gambar m x n pixels
    - Setiap blok dapat dibagi menjadi 4 sub blok
    - Kompleksitas waktu per blok : O(mxn) dengan m dan n adalah ukuran blok
    - Pemrosesan error menentukan apakah blok akan dibagi lagi atau tidak
    - Pada kasus terbaik, blok memenuhi batas threshold sehingga tidak dibagi menjadi 4 bagian lebih kecil → O(mxn)
    - Pada kasus terburuk, blok akan dibagi hingga batas minSize → $O(m \times n \times \log(\min(m, n)/\text{minSize}))$.
    - Kompleksitas waktu dapat dipengaruhi oleh metode pengecekan error, ukuran threshold.
2. Pengaruh Threshold pada kompresi
    - Threshold merupakan batas bawah dalam mengecek warna (R,G,B) pada tiap pixel dalam blok.
    - Semakin kecil threshold, errori blok akan dianggap tidak sama/homogen, sehingga jumlah blok akan semakin banyak
    - Semakin besar threshold, variasi blok akan dianggap sama dan akan dilakukan normalisasi warna rataratanya pada blok tersebut. Hal ini menyebabkan jumlah blok tidak dibagi menjadi lebih kecil lagi.
    - Pengaruh suatu threshold pada tiap error akan berbeda karena memiliki metode/ pendekatan yang berbeda.
    - Ketika target kompresi diisi (!=0) maka ukuran hasil akan disesuaikan dengan cara mencoba menambahkan dan atau mengurangkan threshold dengan nilai tertentu sehingga diharapkan persentase kompresi dapat sedekat mungkin dengan target kompresi.

3. Pengaruh Min Size pada Quadtree
    - MinSize adalah batas minimal pixel dalam blok. Blok tidak akan bisa dibagi jika ukurannya <MinSize
    - Semakin kecil minsize maka gambar hasil kompresi akan semakin detail, namun ukuran file lebih besar, serta waktu kompresi akan semakin lama
    - Semakin besar minSize maka gambar hasil kompresi akan kurang menampilkan detail , namun ukuran relatif lebih kecil dan waktu kompresi lebih singkat.
4. Pengaruh metode pengukuran error yang digunakan
    - Terdapat 4 metode yang dapat digunakan: Variance, Mean Absolute Deviation, Max Pixel Difference , Entropy.
    - Pada besar threshold yang sama, urutan pengaruh dari yang terbesar yaitu metode Max Pixel Difference, Variance, Mean Absolute Deviation, Entropy.Hal ini dibuktikan dengan jumlah simpul yang terbentuk pada threshold (10)dan minsize (8)yang sama: Max Pixel difference (1.387.861), Metode variance (1.338.873), metode MAD(673.101), metode metode entropy (1).

- Metode variance cocok digunakan untuk mendeteksi warna yang cukup sognifikan. Metode ini cocok untuk mempertahankan detail dan memiliki gradasi warna yang cukup halus.
- Metode Mean Absolute deviation mengukur seberapa jauh nilai piksel dari rata ratanya, tetapi tidak memperhatikan arah. Dengan metode ini, waktu lebih efisien dan lebih cepat namun kurang peka terhadap perubahan warna besar yang jumlahnya kecil.
- Metode Max Pixel Difference mengukur kontras maksimal antara piksel tertinggi dengan terendah. Dengan metode ini, waktu yang digunakan sangat efisien dan cukup cepat. Namun kurang cocok pada gambar yang warnanya terdistribusi
- Metode entropy mengukur keragaman informasi warna tiap piksel pada blok. Metode entropy akan lebih cocok pada gambar dengan kompleksitas yang tinggi dan memiliki banyak detail dan halus. Namun metode ini lebih berat dan akan lebih lambat dibanding metode lainnya.

Kesimpulan

Berdasarkan analisis dan percobaan yang telah dilakukan, dapat disimpulkan bahwa algoritma Divide and Conquer dengan pendekatan Quadtree merupakan metode yang efektif untuk melakukan kompresi citra, khususnya pada gambar dengan area warna yang seragam. Algoritma ini bekerja dengan membagi gambar menjadi blok-blok kecil dan menghentikan pembagian jika blok mencapai batas minimal ukuran ataupun batas threshold dari blok tersebut.

Efektivitas kompresi sangat dipengaruhi oleh parameter threshold dan ukuran minimum blok (minSize). Semakin kecil nilai threshold atau minSize, maka semakin tinggi akurasi gambar hasil kompresi, tetapi juga waktu eksekusi akan semakin lama dan ukuran file akan relatif lebih besar. Sebaliknya, nilai threshold dan minSize yang terlalu besar dapat menghasilkan gambar yang sangat terkompresi namun detail gambar mungkin dapat hilang..

Secara keseluruhan, pendekatan Quadtree sangat berguna dalam skenario kompresi gambar yang memerlukan fleksibilitas tinggi dan hasil yang tetap dapat diterima secara visual, terutama bila digabungkan dengan metode evaluasi error yang adaptif dan penyesuaian threshold otomatis berdasarkan target kompresi.

Lampiran

Link Github

https://github.com/StefanMattew/Tucil2_13523020.git

Tabel check

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✅	
2. Program berhasil dijalankan	✅	
3. Program berhasil melakukan kompresi gambar sesuai 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	✅	