

**Tugas Kecil 2 IF2211 Strategi Algoritma
Kompresi Gambar Dengan Metode Quadtree**



Disusun Oleh:
Adiel Rum (10123004)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
TAHUN AKADEMIK 2024/2025**

Daftar Isi

Daftar Isi.....	2
Penjelasan Implementasi dan Pseudocode.....	3
Data Struktur.....	3
Error Measurement.....	4
Class QuadtreeNode.....	5
Class Quadtree.....	8
Fungsi Utama.....	15
Source Program.....	19
Contoh Input dan Output.....	32
Test Case #1.....	32
Test Case #2.....	33
Test Case #3.....	34
Test Case #4.....	35
Test Case #5.....	36
Test Case #6.....	37
Test Case #7.....	38
Analisis Pada Algoritma Kompresi Dengan Quadtree.....	39
1. Analisis Algoritma.....	39
2. Analisis Kompleksitas Waktu.....	39
3. Analisis Kompleksitas Ruang.....	39
4. Faktor yang Mempengaruhi Kinerja.....	40
5. Penurunan Kompleksitas Waktu.....	40
6. Optimasi Potensial.....	40
Implementasi Bonus.....	41
Membuat GIF.....	41
Target Presentase Kompresi.....	42
Lampiran.....	43

Penjelasan Implementasi dan Pseudocode

Data Struktur

Berikut adalah beberapa struktur yang digunakan:

```
STRUCT Config
    DECLARE error_method: INTEGER
    DECLARE error_threshold: FLOAT
    DECLARE min_block_size: INTEGER
    DECLARE target_compression: FLOAT
END STRUCT

STRUCT RGB
    DECLARE r: INTEGER
    DECLARE g: INTEGER
    DECLARE b: INTEGER

    FUNCTION RGB()
        SET r ← 0
        SET g ← 0
        SET b ← 0
    END FUNCTION

    FUNCTION RGB(red, green, blue)
        SET r ← red
        SET g ← green
        SET b ← blue
    END FUNCTION

    FUNCTION operator==(other: RGB) RETURN BOOLEAN
        RETURN (r == other.r) AND (g == other.g) AND (b == other.b)
    END FUNCTION

    FUNCTION operator!=(other: RGB) RETURN BOOLEAN
        RETURN NOT (this == other)
    END FUNCTION
END STRUCT

STRUCT Box
    DECLARE left: INTEGER
    DECLARE top: INTEGER
    DECLARE right: INTEGER
    DECLARE bottom: INTEGER

    // Konstruktor
    FUNCTION Box()
        SET left ← 0
        SET top ← 0
        SET right ← 0
    END FUNCTION
```

```

        SET bottom ← 0
    END FUNCTION

FUNCTION Box(l, t, r, b)
    SET left ← l
    SET top ← t
    SET right ← r
    SET bottom ← b
END FUNCTION

// Fungsi utilitas
FUNCTION width() RETURN INTEGER
    RETURN right - left
END FUNCTION

FUNCTION height() RETURN INTEGER
    RETURN bottom - top
END FUNCTION
END STRUCT

```

Config adalah objek untuk menyimpan parameter-parameter masukan dari pengguna yang akan digunakan dalam proses kompresi, *error_method* adalah metode pengukuran error yang akan digunakan (1 = *Variance*, 2 = *MAD*, 3 = *MaxDiff*, 4 = *SSIM*), *error_threshold* adalah batas ambang batas apakah blok gambar perlu dikompresi lanjut atau tidak, *min_block_size* adalah ukuran minimum blok piksel yang diperbolehkan untuk diproses lanjut, *target_compression* adalah target kompresi

RGB adalah objek untuk menyimpan informasi warna dari warna blok gambar

Box adalah objek untuk menyimpan informasi koordinat dari blok gambar

Error Measurement

Berikut adalah beberapa fungsi yang akan digunakan untuk menghitung error

```

NAMESPACE ErrorMetrics
    // Menghitung Mean Squared Error (MSE) antara gambar asli
    dan terkompresi
    FUNCTION calculate_mse(original: ARRAY OF UNSIGNED BYTE,
                           compressed: ARRAY OF UNSIGNED BYTE,
                           width: INTEGER, height: INTEGER,
                           channels: INTEGER)
        RETURN DOUBLE
        DECLARE sum: DOUBLE ← 0.0
        DECLARE pixels: INTEGER ← width * height
        DECLARE total_samples: INTEGER ← pixels * channels

        FOR i ← 0 TO total_samples - 1 DO
            DECLARE diff: DOUBLE ← original[i] -
            compressed[i]

```

```

        sum ← sum + (diff * diff)
    END FOR

    RETURN sum / total_samples
END FUNCTION

// Menghitung Peak Signal-to-Noise Ratio (PSNR) dari
nilai MSE
FUNCTION calculate_psnr(mse: DOUBLE) RETURN DOUBLE
    IF mse < 0.0001 THEN
        RETURN 100.0
    END IF

    RETURN 10.0 * log10(255.0 * 255.0 / mse)
END FUNCTION

// Menghitung Structural Similarity Index (SSIM)
sederhana
FUNCTION calculate_ssim(original: ARRAY OF UNSIGNED BYTE,
                        compressed: ARRAY OF UNSIGNED
BYTE,
                        width: INTEGER, height: INTEGER,
channels: INTEGER)
    RETURN DOUBLE
    DECLARE mse: DOUBLE ← calculate_mse(original,
compressed, width, height, channels)
    RETURN 1.0 / (1.0 + mse)
END FUNCTION

// Menghitung rasio kompresi
FUNCTION calculate_compression_ratio(original_size:
INTEGER,
                                      compressed_size:
INTEGER)
    RETURN DOUBLE
    RETURN original_size / compressed_size
END FUNCTION
END NAMESPACE

```

Class QuadtreeNode

Berikut adalah class *QuadtreeNode*, termasuk properties, constructor, dan method nya

```

CLASS QuadtreeNode
PUBLIC:
    DECLARE box: Box
    DECLARE depth: INTEGER
    DECLARE color: RGB
    DECLARE error: DOUBLE          // Dihitung
menggunakan salah satu dari empat metode

```

```

DECLARE width: INTEGER
DECLARE height: INTEGER
DECLARE leaf: BOOLEAN           // Menandakan apakah
node ini adalah daun
DECLARE children: LIST OF QuadtreeNode // Urutan:
[kiri-atas, kanan-atas, kiri-bawah, kanan-bawah]

// Konstruktor: menghitung warna rata-rata dan kesalahan
CONSTRUCTOR QuadtreeNode(img_data: ARRAY OF UNSIGNED
BYTE,
                           img_width: INTEGER, img_height:
INTEGER,
                           channels: INTEGER, box: Box,
                           depth: INTEGER,
                           config: Config)
SET this.box ← box
SET this.depth ← depth
SET this.leaf ← FALSE
SET this.width ← box.width()
SET this.height ← box.height()

// Hitung warna rata-rata
DECLARE sum_r, sum_g, sum_b: LONG ← 0
DECLARE count: INTEGER ← 0

FOR y FROM box.top TO box.bottom - 1 DO
    FOR x FROM box.left TO box.right - 1 DO
        DECLARE idx: INTEGER ← (y * img_width + x) *
channels
            sum_r ← sum_r + img_data[idx]
            sum_g ← sum_g + img_data[idx + 1]
            sum_b ← sum_b + img_data[idx + 2]
            count ← count + 1
    END FOR
END FOR

IF count == 0 THEN
    SET count ← 1
END IF

SET color ← RGB(sum_r / count, sum_g / count, sum_b /
count)

// Hitung berbagai metrik kesalahan
DECLARE sumSq: DOUBLE ← 0.0      // Untuk Varians
DECLARE sumAbs: DOUBLE ← 0.0     // Untuk MAD
DECLARE maxDiff: DOUBLE ← 0.0   // Untuk MaksDiff

FOR y FROM box.top TO box.bottom - 1 DO
    FOR x FROM box.left TO box.right - 1 DO

```

```

        DECLARE idx: INTEGER ← (y * img_width + x) *
channels
        DECLARE dr: DOUBLE ← ABS(img_data[idx] -
color.r)
        DECLARE dg: DOUBLE ← ABS(img_data[idx + 1] -
color.g)
        DECLARE db: DOUBLE ← ABS(img_data[idx + 2] -
color.b)

        sumSq ← sumSq + (dr * dr + dg * dg + db * db)
        sumAbs ← sumAbs + (dr + dg + db)
        maxDiff ← MAX(maxDiff, dr, dg, db)
    END FOR
END FOR

DECLARE variance: DOUBLE ← sumSq / (count * 3)
DECLARE mad: DOUBLE ← sumAbs / (count * 3)

// Pilih metode kesalahan berdasarkan konfigurasi
SWITCH config.error_method
    CASE 1: // Varians
        SET error ← variance
    CASE 2: // MAD
        SET error ← mad
    CASE 3: // MaksDiff
        SET error ← maxDiff
    CASE 4: // SSIM
        SET error ← 1.0 / (1.0 + variance)
    DEFAULT:
        SET error ← variance
END SWITCH
END CONSTRUCTOR

// Bagi node menjadi empat kuadran
METHOD split(img_data: ARRAY OF UNSIGNED BYTE,
            img_width: INTEGER, img_height: INTEGER,
            channels: INTEGER, config: Config)

DECLARE midX: INTEGER ← box.left + width / 2
DECLARE midY: INTEGER ← box.top + height / 2

// Buat empat anak node
children.ADD(NEW QuadtreeNode(img_data, img_width,
img_height, channels,
                    Box(box.left, box.top, midX, midY),
depth + 1, config))

children.ADD(NEW QuadtreeNode(img_data, img_width,
img_height, channels,
                    Box(midX, box.top, box.right, midY)),

```

```

depth + 1, config))

    children.ADD(NEW QuadtreeNode(img_data, img_width,
img_height, channels,
                                Box(box.left, midY, midX,
box.bottom), depth + 1, config))

    children.ADD(NEW QuadtreeNode(img_data, img_width,
img_height, channels,
                                Box(midX, midY, box.right,
box.bottom), depth + 1, config))
END METHOD
END CLASS

```

Class ***QuadtreeNode*** memiliki beberapa properties, antara lain:

1. *Box (Box)* : koordinat paga gambar
2. *Depth (int)*: kedalaman node dalam tree
3. *Color (RGB)*: warna rata-rata untuk node tersebut
4. *Error(int)*: metode pengukuran error yang digunakan
5. *Width, height(float)*: panjang dan lebar dari blok gambar
6. *Leaf (boolean)*: menandakan apakah node leaf atau tidak
7. *Children (list)*: list yang menyimpan node children (jika ada)

QuadtreeNode dibangun dengan constructor:

1. Menghitung nilai rata-rata rgb dari blok gambar
2. Menghitung nilai error untuk blok gambar
3. Jika nilai error masih lebih besar dari threshold, maka akan split

QuadtreeNode memiliki metode *split* untuk memecah gambar menjadi 4 node dengan cara:

1. Menghitung titik tengah dari kotak
2. Membagi menjadi empat anak lebih kecil
3. Construct 4 node baru dari yang sudah dibagi

Class Quadtree

```

CLASS Quadtree
PRIVATE:
    DECLARE root: POINTER TO QuadtreeNode
    DECLARE width, height: INTEGER
    DECLARE max_depth: INTEGER
    DECLARE original_data: ARRAY OF UNSIGNED BYTE
    DECLARE img_data: ARRAY OF UNSIGNED BYTE // pointer
ke data gambar asli
    DECLARE channels: INTEGER
    DECLARE config: Config

    // Membangun quadtree secara rekursif
METHOD build_tree(img_data: ARRAY OF UNSIGNED BYTE,
                  img_width: INTEGER, img_height:
INTEGER,

```



```

OF POINTER TO QuadtreeNode)
    IF node->leaf OR node->depth == target_depth THEN
        result.ADD(node)
    ELSE
        FOR EACH child IN node->children DO
            CALL get_leaf_nodes_recursion(child,
target_depth, result)
        END FOR
    END IF
END METHOD

// Hitung semua node dalam pohon
METHOD count_all_nodes(node: POINTER TO QuadtreeNode)
RETURN INTEGER
    DECLARE count: INTEGER ← 1
    FOR EACH child IN node->children DO
        count ← count + CALL count_all_nodes(child)
    END FOR
    RETURN count
END METHOD

// Buat buffer gambar dari quadtree pada kedalaman tertentu
METHOD create_image_from_depth(depth: INTEGER) RETURN
ARRAY OF UNSIGNED BYTE
    DECLARE output_data: ARRAY OF UNSIGNED BYTE ← NEW
ARRAY[width * height * 3]
    FILL output_data WITH 0

    DECLARE leaf_nodes: LIST OF POINTER TO
QuadtreeNode
    CALL get_leaf_nodes_recursion(root, depth,
leaf_nodes)

    FOR EACH node IN leaf_nodes DO
        FOR y FROM node->box.top TO node->box.bottom
- 1 DO
            FOR x FROM node->box.left TO
node->box.right - 1 DO
                DECLARE idx: INTEGER ← (y * width +
x) * 3
                output_data[idx] ← node->color.r
                output_data[idx + 1] ← node->color.g
                output_data[idx + 2] ← node->color.b
            END FOR
        END FOR
    END FOR

    RETURN output_data
END METHOD

```

```

PUBLIC:
    // Konstruktor membangun quadtree dari data gambar
    CONSTRUCTOR Quadtree(img_data: ARRAY OF UNSIGNED
BYTE,
                        img_width: INTEGER, img_height:
INTEGER,
                        channels: INTEGER, cfg: Config)
        SET width ← img_width
        SET height ← img_height
        SET channels ← channels
        SET config ← cfg
        SET max_depth ← 0

        // Salin data gambar asli
        SET original_data ← NEW ARRAY[width * height * channels]
        COPY img_data TO original_data
        SET this->img_data ← img_data

        // Buat root node
        DECLARE full_img: Box ← Box(0, 0, width, height)
        SET root ← NEW QuadTreeNode(img_data, img_width,
img_height, channels, full_img, 0, config)

        // Bangun pohon
        CALL build_tree(img_data, img_width, img_height,
channels, root)
    END CONSTRUCTOR

    DESTRUCTOR Quadtree()
        DELETE original_data
    END DESTRUCTOR

    // Dapatkan node daun pada kedalaman tertentu
    METHOD get_leaf_nodes(depth: INTEGER) RETURN LIST OF
POINTER TO QuadTreeNode
        IF depth > max_depth THEN
            THROW ERROR "Kedalaman melebihi maksimum"
        END IF

        DECLARE leaf_nodes: LIST OF POINTER TO
QuadTreeNode
        CALL get_leaf_nodes_recursion(root, depth,
leaf_nodes)
        RETURN leaf_nodes
    END METHOD

    // Render dan simpan gambar terkompresi
    METHOD render_at_depth(depth: INTEGER,

```

```

output_filename: STRING)
    IF depth > max_depth THEN
        THROW ERROR "Kedalaman melebihi maksimum"
    END IF

        DECLARE output_data: ARRAY OF UNSIGNED BYTE ←
CALL create_image_from_depth(depth)
        SAVE output_data AS PNG TO output_filename

        // Hitung metrik kesalahan
        DECLARE mse: DOUBLE ←
ErrorMetrics.calculate_mse(original_data, output_data, width,
height, 3)
        DECLARE psnr: DOUBLE ←
ErrorMetrics.calculate_psnr(mse)
        DECLARE ssim: DOUBLE ←
ErrorMetrics.calculate_ssim(original_data, output_data,
width, height, 3)

        DECLARE total_leaf_nodes: INTEGER ← CALL
get_leaf_nodes(depth).SIZE()
        DECLARE original_size: INTEGER ← width * height *
3
        DECLARE estimated_size: INTEGER ←
total_leaf_nodes * (SIZEOF(RGB) + SIZEOF(Box))
        DECLARE comp_ratio: DOUBLE ←
ErrorMetrics.calculate_compression_ratio(original_size,
estimated_size)

        PRINT "--- Metrik kesalahan pada kedalaman " +
depth + " ---"
        PRINT "Metode kesalahan: " +
GET_METHOD_NAME(config.error_method)
        PRINT "MSE: " + mse
        PRINT "PSNR: " + psnr + " dB"
        PRINT "SSIM: " + ssim
        PRINT "Ratio kompresi: " + comp_ratio + ":1"
        PRINT "Total node daun: " + total_leaf_nodes

        DELETE output_data
END METHOD

// Buat GIF animasi proses kompresi
METHOD create_gif(filename: STRING, delay_ms: INTEGER
= 100)
    INITIALIZE GIF WRITER WITH filename, width,
height

    FOR d FROM 0 TO max_depth DO
        DECLARE frame: ARRAY OF UNSIGNED BYTE ← CALL

```

```

create_image_from_depth(d)
    CONVERT frame TO RGBA
    ADD FRAME TO GIF WITH delay_ms/10
    DELETE frame
END FOR

// Tambahkan frame akhir beberapa kali
DECLARE final_frame: ARRAY OF UNSIGNED BYTE ←
CALL create_image_from_depth(max_depth)
    CONVERT final_frame TO RGBA
    REPEAT 4 TIMES:
        ADD final_frame TO GIF WITH delay_ms/10

FINALIZE GIF
PRINT "GIF disimpan di " + filename
DELETE final_frame
END METHOD

// Analisis kesalahan di semua kedalaman
METHOD analyze_all_depths()
    PRINT "--- Analisis Kesalahan di Semua Kedalaman
---"
    PRINT "Kedalaman | MSE      | PSNR (dB) | SSIM
| Node Daun | Rasio Kompresi"
    PRINT
"-----"
"---"

FOR d FROM 0 TO max_depth DO
    DECLARE comp: ARRAY OF UNSIGNED BYTE ← CALL
create_image_from_depth(d)
    DECLARE mse: DOUBLE ←
ErrorMetrics.calculate_mse(original_data, comp, width,
height, 3)
    DECLARE psnr: DOUBLE ←
ErrorMetrics.calculate_psnr(mse)
    DECLARE ssim: DOUBLE ←
ErrorMetrics.calculate_ssim(original_data, comp, width,
height, 3)
    DECLARE leafCount: INTEGER ← CALL
get_leaf_nodes(d).SIZE()
    DECLARE origSize: INTEGER ← width * height *
3
    DECLARE estSize: INTEGER ← leafCount *
(SIZEOF(RGB) + SIZEOF(Box))
    DECLARE ratio: DOUBLE ←
ErrorMetrics.calculate_compression_ratio(origSize, estSize)

    PRINT FORMAT("%9d | %7.2f | %9.2f | %7.4f |
%9d | %10.2f:1", d, mse, psnr, ssim, leafCount, ratio)

```

```

        DELETE comp
    END FOR
END METHOD

METHOD get_max_depth() RETURN INTEGER
    RETURN max_depth
END METHOD

METHOD get_total_nodes() RETURN INTEGER
    RETURN CALL count_all_nodes(root)
END METHOD
END CLASS

```

Class ***Quadtree*** memiliki beberapa properties:

1. *Root (QuadtreeNode)*: root dari quadtree untuk gambar kesluruhan
2. *Width, height*: dimensi dari gambar asli
3. *Channels (int)*: banyak channel (asumsi 3: *rgb*)
4. *Original_data*: gambar orisinil untuk menghitung error
5. *Img_data (pointer)*: pointer yang mengarah ke gambar terproses
6. *Config (config)*: config parameter masukan pengguna
7. *Max_depth (int)*: kedalaman maksimum pada quadtree

Quadtree dibangun oleh constructor berikut:

1. Simpan salinan asli dari gambar
2. Buat root dari *QuadtreeNode* sebagai root dari gambar dengan *Box(0,0,)*
3. Panggil metode *build_tree()* mulai dari root untuk membangun quadtree

Quadtree memiliki beberapa metode untuk membantu proses kompresi

Build_tree() adalah metode untuk membangun quadtree:

1. Jika node sudah lebih kecil dari *min_block_size*, maka tandai node menjadi leaf
2. Tentukan apakah node sudah memenuhi kriteria threshold
3. Jika sudah, maka tandai node sebagai leaf dan update *max_depth*
4. Jika belum, panggil metode *node.split()* untuk melakukan kompresi
5. Panggil secara rekursif metode *build_tree()* pada masing-masing node anak

get_leaf_nodes(): mencari secara rekursif semua node pada kedalaman tertentu

create_image_at_depth(): membuat gambar untuk kedalaman tertentu dengan cara:

1. Membuat buffer gambar
2. Ambil node pada kedalaman tertentu dengan *get_leaf_nodes()*
3. Untuk setiap node, isi piksel pada buffer dengan warna rata-rata
4. Mengembalikan buffer gambar

render_at_depth(): memproses gambar untuk kedalaman tertentu

1. Panggil *create_image_at_depth()* untuk menghasilkan gambar terkompresi
2. Simpan gambar dengan fungsi write
3. Hitung error untuk pelaporan
4. Hitung estimasi kompresi
5. Print metrik error dan data untuk node

create_gif(): fungsi untuk membuat gif dengan memproses gambar pada setiap iterasi
get_total_nodes(): fungsi rekursif untuk mendapatkan total nodes

Fungsi Utama

Berikut adalah fungsi utama dari program ini yang melakukan kompresi gambar

```
FUNCTION main() RETURN INTEGER
    // 1. Input path gambar
    PRINT "Masukkan path absolut gambar yang akan dikompresi:
"
    DECLARE input_image_path: STRING
    GETLINE input_image_path FROM USER

    // 2. Pilih metode pengukuran kesalahan
    PRINT "Masukkan metode pengukuran kesalahan (1 = Varians,
2 = MAD, 3 = MaksDiff, 4 = SSIM): "
    DECLARE method: INTEGER
    GET method FROM USER

    // 3. Input ambang batas kesalahan
    PRINT "Masukkan ambang batas kesalahan:"
    PRINT " - Untuk Varians, coba nilai antara 10 dan 50."
    PRINT " - Untuk MAD, coba nilai antara 5 dan 20."
    PRINT " - Untuk MaksDiff, coba nilai antara 20 dan 100."
    PRINT " - Untuk SSIM, gunakan nilai antara 0 dan 1
(misal, 0,90 sampai 0,99)."
    PRINT "Input Anda: "
    DECLARE threshold: FLOAT
    GET threshold FROM USER

    // 4. Input ukuran blok minimum
    PRINT "Masukkan ukuran blok minimum: "
    DECLARE min_block: INTEGER
    GET min_block FROM USER

    // 5. Input rasio kompresi target
    PRINT "Masukkan rasio kompresi target (misal, 2.0 untuk
kompresi 2:1, 0 untuk menonaktifkan): "
    DECLARE targetCompression: FLOAT
    GET targetCompression FROM USER
    CLEAR INPUT BUFFER

    // 6. Input path output gambar
    PRINT "Masukkan path absolut untuk output gambar
terkompresi: "
    DECLARE output_image_path: STRING
    GETLINE output_image_path FROM USER

    // 7. Input path output GIF (opsional)
    PRINT "Masukkan path absolut untuk output GIF (kosongkan
```

```

untuk melewati): "
DECLARE output_gif_path: STRING
GETLINE output_gif_path FROM USER

// Muat gambar input
DECLARE width, height, channels: INTEGER
DECLARE img_data: ARRAY OF UNSIGNED BYTE ←
LOAD_IMAGE(input_image_path, REF width, REF height, REF
channels)
IF img_data IS NULL THEN
    PRINT "Error: Tidak dapat memuat gambar " +
input_image_path
    RETURN 1
END IF
SET channels ← 3
PRINT "Gambar dimuat: " + width + "x" + height + ", " +
channels + " saluran"

// Catat waktu mulai
DECLARE t_start: TIME ← GET_CURRENT_TIME()

// Siapkan konfigurasi
DECLARE config: Config
SET config.error_method ← method
SET config.error_threshold ← threshold
SET config.min_block_size ← min_block
SET config.target_compression ← targetCompression

// Bangun quadtree
DECLARE tree: Quadtree ← NEW Quadtree(img_data, width,
height, channels, config)

// Penyesuaian otomatis ambang batas (untuk metode
Varians)
IF config.target_compression > 0 AND config.error_method
== 1 THEN
    DECLARE low: FLOAT ← 0.0
    DECLARE high: FLOAT ← 255.0
    DECLARE iterations: INTEGER ← 15

    FOR i FROM 1 TO iterations DO
        SET config.error_threshold ← (low + high) / 2.0
        DECLARE tempTree: Quadtree ← NEW
Quadtree(img_data, width, height, channels, config)
        DECLARE leafCount: INTEGER ← CALL
tempTree.get_leaf_nodes(tempTree.get_max_depth()).SIZE()
        DECLARE origSize: INTEGER ← width * height * 3
        DECLARE estSize: INTEGER ← leafCount *
(SIZEOF(RGB) + SIZEOF(Box))
        DECLARE achievedRatio: FLOAT ← origSize / estSize

```

```

        IF achievedRatio < config.target_compression THEN
            SET low ← config.error_threshold
        ELSE
            SET high ← config.error_threshold
        END IF
    END FOR

    PRINT "Ambang batas Varians disesuaikan otomatis
menjadi: " + config.error_threshold
    SET tree ← NEW Quadtree(img_data, width, height,
channels, config)
END IF

DECLARE render_depth: INTEGER ← CALL tree.get_max_depth()

// Simpan gambar terkompresi
CALL tree.render_at_depth(render_depth,
output_image_path)
PRINT "Gambar terkompresi disimpan di: " +
output_image_path

// Buat GIF jika path disediakan
IF output_gif_path IS NOT EMPTY THEN
    CALL tree.create_gif(output_gif_path, 500)
END IF

// Hitung statistik kompresi
DECLARE total_nodes: INTEGER ← CALL
tree.get_total_nodes()
DECLARE leafCount: INTEGER ← CALL
tree.get_leaf_nodes(render_depth).SIZE()
DECLARE original_size: INTEGER ← width * height * 3
DECLARE estimated_size: INTEGER ← leafCount *
(SIZEOF(RGB) + SIZEOF(Box))
DECLARE comp_ratio: FLOAT ← original_size /
estimated_size
DECLARE comp_percentage: FLOAT ← (1.0 - estimated_size /
original_size) * 100.0

DECLARE t_end: TIME ← GET_CURRENT_TIME()
DECLARE elapsed: FLOAT ← t_end - t_start

// Tampilkan laporan kompresi
PRINT "\n---- Laporan Kompresi ----"
PRINT "Waktu eksekusi: " + elapsed + " detik"
PRINT "Ukuran gambar asli: " + original_size + " byte"
PRINT "Perkiraan ukuran terkompresi: " + estimated_size +
" byte"
PRINT "Persentase kompresi: " + comp_percentage + " %"

```

```

PRINT "Kedalaman quadtree: " + tree.get_max_depth()
PRINT "Jumlah total node: " + total_nodes

// Bersihkan memori
FREE img_data
RETURN 0
END FUNCTION

```

Fungsi utama program ini memiliki alur sebagai berikut

1. Minta pengguna untuk memasukkan input:
 - a. Path absolut untuk gambar input.
 - b. Metode pengukuran error (1 = Variance, 2 = MAD, 3 = MaxDiff, 4 = SSIM).
 - c. Threshold error (disertai saran nilai yang baik).
 - d. Ukuran blok minimum.
 - e. Rasio kompresi target (masukkan 0 untuk menonaktifkan penyesuaian otomatis).
 - f. Path absolut output untuk gambar terkompresi.
 - g. Path absolut output untuk GIF (input kosong berarti lewati pembuatan GIF).
2. Muat gambar menggunakan path yang diberikan. Pastikan gambar berhasil dimuat dengan benar.
3. Mulai timer untuk mengukur waktu eksekusi.
4. Buat struktur Config dengan parameter input.
5. Buat objek Quadtree dengan memasukkan data gambar, dimensi, saluran warna, dan Config.
6. (Bonus) Jika rasio kompresi target diaktifkan dan error_method adalah Variance; Gunakan binary search untuk menyesuaikan error_threshold sehingga rasio kompresi yang diestimasi memenuhi target. Bangun ulang quadtree dengan threshold yang telah disesuaikan.
7. Gunakan kedalaman maksimum quadtree sebagai kedalaman rendering.
8. Panggil render_at_depth() untuk menghasilkan dan menyimpan gambar terkompresi.
9. Jika path output GIF diberikan, panggil create_gif() untuk membuat animasi kompresi.
10. Dapatkan dan cetak metrik laporan berikut:
 - Waktu eksekusi.
 - Ukuran gambar asli (dalam byte).
 - Ukuran terkompresi yang diestimasi (dalam byte).
 - Persentase kompresi.
 - Kedalaman akhir quadtree.
 - Jumlah total node.
 - Analisis error di semua kedalaman.
11. Bebaskan memori gambar yang dialokasikan.

Source Program

Berikut adalah *source code* untuk program kompresi gambar dengan metode Quadtree dengan bahasa yang dipilih adalah C++. Library yang digunakan adalah C++ standard library dan stb_image library untuk pemrosesan gambar

```
#include <bits/stdc++.h>
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"
#include "gif.h"

using namespace std;

// ===== Konfigurasi =====
struct Config {
    int error_method;           // 1 = Varians, 2 = MAD, 3 = MaksDiff,
4 = SSIM
    float error_threshold;      // nilai batas kesalahan
    int min_block_size;         // Ukuran blok minimum sebelum memaksa
menjadi daun
    float target_compression;   // Rasio kompresi target (ukuran_asli
/ perkiraan_ukuran_terkompresi)
                                // 0 untuk menonaktifkan penyesuaian
otomatis
};

// --- Catatan parameter yang baik untuk nilai batas kesalahan ---
// Varians: Gunakan angka kecil (misalnya 10 sampai 50)
// MAD: Nilai tipikal sekitar 5 sampai 20
// MaksDiff: Batas yang berguna sekitar 20 sampai 100
// SSIM: Karena nilainya dalam (0,1], ambang batas seperti 0,90
sampai 0,99 bekerja dengan baik

// Struktur RGB
struct RGB {
    int r, g, b;
    RGB() : r(0), g(0), b(0) {}
    RGB(int r, int g, int b) : r(r), g(g), b(b) {}
    bool operator==(const RGB& other) const { return r == other.r &&
g == other.g && b == other.b; }
    bool operator!=(const RGB& other) const { return !(*this ==
other); }
```

```
};

// Struktur Box untuk merepresentasikan area persegi panjang dalam
gambar
struct Box {
    int left, top, right, bottom;
    Box() : left(0), top(0), right(0), bottom(0) {}
    Box(int l, int t, int r, int b) : left(l), top(t), right(r),
bottom(b) {}
    int width() const { return right - left; }
    int height() const { return bottom - top; }
};

// Metrik Kesalahan untuk pelaporan
namespace ErrorMetrics {
    double calculate_mse(unsigned char* original, unsigned char*
compressed, int width, int height, int channels) {
        double sum = 0.0;
        int pixels = width * height;
        int total_samples = pixels * channels;
        for (int i = 0; i < total_samples; i++) {
            double diff = original[i] - compressed[i];
            sum += diff * diff;
        }
        return sum / total_samples;
    }

    double calculate_psnr(double mse) {
        if (mse < 0.0001) return 100.0;
        return 10.0 * log10(255.0 * 255.0 / mse);
    }

    double calculate_ssim(unsigned char* original, unsigned char*
compressed, int width, int height, int channels) {
        double mse = calculate_mse(original, compressed, width,
height, channels);
        return 1.0 / (1.0 + mse);
    }

    double calculate_compression_ratio(int original_size, int
compressed_size) {
        return static_cast<double>(original_size) / compressed_size;
    }
}
```

```
    }

};

// ===== Kelas Node Quadtree =====
class QuadTreeNode {
public:
    Box box;
    int depth;
    RGB color;
    // 'error' dihitung menggunakan salah satu dari empat metode
    double error;
    int width, height;
    bool leaf;
    vector<shared_ptr<QuadTreeNode>> children; // Urutan: kiri-atas,
    kanan-atas, kiri-bawah, kanan-bawah

    // Konstruktor: menghitung warna rata-rata dan kesalahan
    QuadTreeNode(unsigned char* img_data, int img_width, int
    img_height, int channels,
                 const Box& box, int depth, const Config &config)
        : box(box), depth(depth), leaf(false)
    {
        width = box.width();
        height = box.height();

        // Hitung warna rata-rata
        long sum_r = 0, sum_g = 0, sum_b = 0;
        int count = 0;
        for (int y = box.top; y < box.bottom; y++) {
            for (int x = box.left; x < box.right; x++) {
                int idx = (y * img_width + x) * channels;
                sum_r += img_data[idx];
                sum_g += img_data[idx + 1];
                sum_b += img_data[idx + 2];
                count++;
            }
        }
        if (count == 0) count = 1;
        color = RGB(static_cast<int>(sum_r / count),
                    static_cast<int>(sum_g / count),
                    static_cast<int>(sum_b / count));
    }
}
```

```

// Untuk setiap piksel, hitung perbedaan dari rata-rata
    double sumSq = 0.0; // Untuk Varians (perbedaan kuadrat
rata-rata)

    double sumAbs = 0.0; // Untuk MAD (deviasi absolut rata-rata)
    double maxDiff = 0.0; // Untuk MaksDiff
    for (int y = box.top; y < box.bottom; y++) {
        for (int x = box.left; x < box.right; x++) {
            int idx = (y * img_width + x) * channels;
            double dr = abs(img_data[idx] - color.r);
            double dg = abs(img_data[idx + 1] - color.g);
            double db = abs(img_data[idx + 2] - color.b);
            sumSq += (dr * dr + dg * dg + db * db);
            sumAbs += (dr + dg + db);
            maxDiff = max({maxDiff, dr, dg, db});
        }
    }
    double variance = sumSq / (count * 3);
    double mad = sumAbs / (count * 3);

    // Hitung kesalahan berdasarkan metode yang dipilih
    if (config.error_method == 1) {
        error = variance;
    } else if (config.error_method == 2) {
        error = mad;
    } else if (config.error_method == 3) {
        error = maxDiff;
    } else if (config.error_method == 4) {
        error = 1.0 / (1.0 + variance);
    } else {
        error = variance; // Default
    }
}

// Bagi node menjadi empat kuadran
void split(unsigned char* img_data, int img_width, int
img_height, int channels, const Config &config) {
    int midX = box.left + width / 2;
    int midY = box.top + height / 2;
    children.push_back(make_shared<QuadtreeNode>(img_data,
img_width, img_height, channels,
                                         Box(box.left, box.top, midX,
midY), depth + 1, config));
}

```

```

        children.push_back(make_shared<QuadtreeNode>(img_data,
img_width, img_height, channels,
                                         Box(midX, box.top, box.right,
midY), depth + 1, config));
        children.push_back(make_shared<QuadtreeNode>(img_data,
img_width, img_height, channels,
                                         Box(box.left, midY, midX,
box.bottom), depth + 1, config));
        children.push_back(make_shared<QuadtreeNode>(img_data,
img_width, img_height, channels,
                                         Box(midX, midY, box.right,
box.bottom), depth + 1, config));
    }
};

// ===== Kelas Quadtree =====
class Quadtree {
private:
    shared_ptr<QuadtreeNode> root;
    int width, height;
    int max_depth;
    unsigned char* original_data;
    unsigned char* img_data; // pointer ke data gambar asli
    int channels;
    Config config;

    // Membangun quadtree secara rekursif
    void build_tree(unsigned char* img_data, int img_width, int
img_height, int channels,
                    shared_ptr<QuadtreeNode> node) {
        // Berhenti membagi jika blok lebih kecil dari minimum yang
diizinkan
        if (node->width <= config.min_block_size || node->height <=
config.min_block_size) {
            node->leaf = true;
            if (node->depth > max_depth)
                max_depth = node->depth;
            return;
        }

        // Tentukan kondisi berhenti berdasarkan metode kesalahan
        bool stopSplit = false;

```

```

    if (config.error_method == 4) {
        // Untuk SSIM, semakin tinggi semakin baik
        if (node->error >= config.error_threshold)
            stopSplit = true;
    } else {
        // Untuk Varians, MAD, dan MaksDiff lebih rendah lebih
baik
        if (node->error <= config.error_threshold)
            stopSplit = true;
    }

    if (stopSplit) {
        node->leaf = true;
        if (node->depth > max_depth)
            max_depth = node->depth;
        return;
    }

    // Jika tidak, bagi node
    node->split(img_data, img_width, img_height, channels,
config);
    for (auto &child : node->children) {
        build_tree(img_data, img_width, img_height, channels,
child);
    }
}

// Kumpulkan node daun pada atau di bawah kedalaman tertentu
void get_leaf_nodes_recursion(shared_ptr<QuadtreeNode> node, int
target_depth,
                                vector<shared_ptr<QuadtreeNode>>
&result) {
    if (node->leaf || node->depth == target_depth) {
        result.push_back(node);
    } else {
        for (auto &child : node->children)
            get_leaf_nodes_recursion(child, target_depth,
result);
    }
}

// Hitung semua node dalam pohon

```

```

int count_all_nodes(shared_ptr<QuadtreeNode> node) {
    int count = 1;
    for (auto &child : node->children)
        count += count_all_nodes(child);
    return count;
}

// Buat buffer gambar yang mewakili perkiraan quadtree pada
// kedalaman tertentu
unsigned char* create_image_from_depth(int depth) {
    unsigned char* output_data = new unsigned char[width * height
* 3];
    fill(output_data, output_data + width * height * 3, 0);
    vector<shared_ptr<QuadtreeNode>> leaf_nodes;
    get_leaf_nodes_recursion(root, depth, leaf_nodes);
    for (const auto &node : leaf_nodes) {
        for (int y = node->box.top; y < node->box.bottom; y++) {
            for (int x = node->box.left; x < node->box.right;
x++) {
                int idx = (y * width + x) * 3;
                output_data[idx] = node->color.r;
                output_data[idx + 1] = node->color.g;
                output_data[idx + 2] = node->color.b;
            }
        }
    }
    return output_data;
}

public:
    // Konstruktor membangun quadtree dari data gambar dengan
    // konfigurasi yang diberikan
    Quadtree(unsigned char* img_data, int img_width, int img_height,
int channels, const Config &cfg)
        : width(img_width), height(img_height), channels(channels),
config(cfg), max_depth(0)
    {
        original_data = new unsigned char[width * height * channels];
        copy(img_data, img_data + width * height * channels,
original_data);
        this->img_data = img_data;
        Box full_img(0, 0, width, height);

```

```

        root = make_shared<QuadtreeNode>(img_data, img_width,
img_height, channels, full_img, 0, config);
    build_tree(img_data, img_width, img_height, channels, root);
}

~Quadtree() { delete[] original_data; }

vector<shared_ptr<QuadtreeNode>> get_leaf_nodes(int depth) {
    if (depth > max_depth)
        throw invalid_argument("Kedalaman yang diminta melebihi
kedalaman pohon");
    vector<shared_ptr<QuadtreeNode>> leaf_nodes;
    get_leaf_nodes_recursion(root, depth, leaf_nodes);
    return leaf_nodes;
}

// Render dan simpan gambar terkompresi pada kedalaman tertentu
void render_at_depth(int depth, const string &output_filename) {
    if (depth > max_depth)
        throw invalid_argument("Kedalaman yang diminta melebihi
kedalaman pohon");
    unsigned char* output_data = create_image_from_depth(depth);
    stbi_write_png(output_filename.c_str(), width, height, 3,
output_data, width * 3);

    // Hitung metrik kesalahan gambar penuh untuk pelaporan
    double mse = ErrorMetrics::calculate_mse(original_data,
output_data, width, height, 3);
    double psnr = ErrorMetrics::calculate_psnr(mse);
    double ssim = ErrorMetrics::calculate_ssim(original_data,
output_data, width, height, 3);

    int total_leaf_nodes = get_leaf_nodes(depth).size();
    int original_size = width * height * 3;
    int estimated_size = total_leaf_nodes * (sizeof(RGB) +
sizeof(Box));
    double comp_ratio =
ErrorMetrics::calculate_compression_ratio(original_size,
estimated_size);

    cout << "\n--- Metrik kesalahan pada kedalaman " << depth <<
" ---\n";
}

```

```

cout << "Menggunakan metode kesalahan: ";
if (config.error_method == 1) cout << "Varians";
else if (config.error_method == 2) cout << "MAD";
else if (config.error_method == 3) cout << "MaksDiff";
else if (config.error_method == 4) cout << "SSIM";
cout << "\nMSE gambar penuh: " << mse;
cout << "\nPSNR gambar penuh: " << psnr << " dB";
cout << "\nSSIM gambar penuh: " << ssim;
cout << "\nPerkiraan rasio kompresi: " << comp_ratio << ":1";
cout << "\nTotal node daun: " << total_leaf_nodes << "\n";

delete[] output_data;
}

// Buat GIF yang menunjukkan proses kompresi
void create_gif(const string &filename, int delay_ms = 100) {
    GifWriter g;
    GifBegin(&g, filename.c_str(), width, height, delay_ms / 10);
    for (int d = 0; d <= max_depth; d++) {
        unsigned char* frame = create_image_from_depth(d);
        uint8_t* rgba = new uint8_t[width * height * 4];
        for (int i = 0; i < width * height; i++) {
            rgba[i * 4 + 0] = frame[i * 3 + 0];
            rgba[i * 4 + 1] = frame[i * 3 + 1];
            rgba[i * 4 + 2] = frame[i * 3 + 2];
            rgba[i * 4 + 3] = 255;
        }
        GifWriteFrame(&g, rgba, width, height, delay_ms / 10);
        delete[] rgba;
        delete[] frame;
    }
    unsigned char* final_frame =
    create_image_from_depth(max_depth);
    uint8_t* final_rgba = new uint8_t[width * height * 4];
    for (int i = 0; i < width * height; i++) {
        final_rgba[i * 4 + 0] = final_frame[i * 3 + 0];
        final_rgba[i * 4 + 1] = final_frame[i * 3 + 1];
        final_rgba[i * 4 + 2] = final_frame[i * 3 + 2];
        final_rgba[i * 4 + 3] = 255;
    }
    for (int i = 0; i < 4; i++)
        GifWriteFrame(&g, final_rgba, width, height, delay_ms /

```

```

10);
    GifEnd(&g);
    delete[] final_rgba;
    delete[] final_frame;
    cout << "GIF telah dibuat di " << filename << "\n";
}

int get_max_depth() const { return max_depth; }
int get_total_nodes() { return count_all_nodes(root); }
};

// ===== Fungsi Utama =====
int main() {
    // 1. Path absolut gambar input
    string input_image_path;
    cout << "Masukkan path absolut gambar yang akan dikompresi: ";
    getline(cin, input_image_path);

    // 2. Metode pengukuran kesalahan
    int method;
    cout << "Masukkan metode pengukuran kesalahan (1 = Varians, 2 =
MAD, 3 = MaksDiff, 4 = SSIM): ";
    cin >> method;

    // 3. Ambang batas kesalahan
    float threshold;
    cout << "Masukkan ambang batas kesalahan:\n";
    cout << " - Untuk Varians, coba nilai antara 10 dan 50.\n";
    cout << " - Untuk MAD, coba nilai antara 5 dan 20.\n";
    cout << " - Untuk MaksDiff, coba nilai antara 20 dan 100.\n";
    cout << " - Untuk SSIM, gunakan nilai antara 0 dan 1 (misal, 0,90
sampai 0,99).\n";
    cout << "Input Anda: ";
    cin >> threshold;

    // 4. Ukuran blok minimum
    int min_block;
    cout << "Masukkan ukuran blok minimum: ";
    cin >> min_block;

    // 5. Rasio kompresi target
}

```

```
float targetCompression;
cout << "Masukkan rasio kompresi target (misal, 2.0 untuk
kompresi 2:1, 0 untuk menonaktifkan penyesuaian otomatis): ";
cin >> targetCompression;
cin.ignore(); // Bersihkan newline

// 6. Path absolut gambar terkompresi output
string output_image_path;
cout << "Masukkan path absolut untuk output gambar terkompresi:
";
getline(cin, output_image_path);

// 7. Path absolut GIF output (jika kosong, GIF dilewati)
string output_gif_path;
cout << "Masukkan path absolut untuk output GIF (kosongkan untuk
melewati): ";
getline(cin, output_gif_path);

// Muat gambar input
int width, height, channels;
unsigned char* img_data = stbi_load(input_image_path.c_str(),
&width, &height, &channels, 3);
if (!img_data) {
    cerr << "Error: Tidak dapat memuat gambar " <<
input_image_path << "\n";
    return 1;
}
channels = 3;
cout << "Gambar dimuat: " << width << "x" << height << ", " <<
channels << " saluran\n";

// Mulai timer
auto t_start = chrono::high_resolution_clock::now();

// Siapkan konfigurasi
Config config;
config.error_method = method;
config.error_threshold = threshold;
config.min_block_size = min_block;
config.target_compression = targetCompression;

// Bangun quadtree
```

```

Quadtree tree(img_data, width, height, channels, config);

// Penyesuaian otomatis ambang batas (hanya untuk metode Varians)
if (config.target_compression > 0 && config.error_method == 1) {
    float low = 0.0f, high = 255.0f;
    const int iterations = 15;
    for (int i = 0; i < iterations; i++) {
        config.error_threshold = (low + high) / 2.0f;
        Quadtree tempTree(img_data, width, height, channels,
config);
        int leafCount =
tempTree.get_leaf_nodes(tempTree.get_max_depth()).size();
        int origSize = width * height * 3;
        int estSize = leafCount * (sizeof(RGB) + sizeof(Box));
        double achievedRatio = static_cast<double>(origSize) /
estSize;
        if (achievedRatio < config.target_compression)
            low = config.error_threshold;
        else
            high = config.error_threshold;
    }
    cout << "Ambang batas Varians disesuaikan otomatis menjadi: "
<< config.error_threshold << "\n";
    tree = Quadtree(img_data, width, height, channels, config);
}

int render_depth = tree.get_max_depth();

// Render dan simpan gambar terkompresi
tree.render_at_depth(render_depth, output_image_path);
cout << "Gambar terkompresi disimpan di: " << output_image_path
<< "\n";

// Buat GIF jika path disediakan
if (!output_gif_path.empty()) {
    tree.create_gif(output_gif_path, 500);
}

int total_nodes = tree.get_total_nodes();
int leafCount = tree.get_leaf_nodes(render_depth).size();
int original_size = width * height * 3;
int estimated_size = leafCount * (sizeof(RGB) + sizeof(Box));

```

```
    double comp_ratio = static_cast<double>(original_size) /
estimated_size;
    double comp_percentage = (1.0 -
static_cast<double>(estimated_size) / original_size) * 100.0;

    auto t_end = chrono::high_resolution_clock::now();
chrono::duration<double> elapsed = t_end - t_start;

cout << "\n----- Laporan Kompresi ----- \n";
cout << "Waktu eksekusi: " << elapsed.count() << " detik\n";
cout << "Ukuran gambar asli: " << original_size << " byte\n";
cout << "Perkiraan ukuran terkompresi: " << estimated_size << "
byte\n";
cout << "Persentase kompresi: " << comp_percentage << " %\n";
cout << "Kedalaman quadtree: " << tree.get_max_depth() << "\n";
cout << "Jumlah total node: " << total_nodes << "\n";

stbi_image_free(img_data);
return 0;
}
```

Contoh Input dan Output

Test Case #1

```
Masukkan path absolut gambar yang akan dikompresi: D:\Projects\stima\Tucil2-10123004\test\original\test1.jpg
Masukkan metode pengukuran kesalahan (1 = Varians, 2 = MAD, 3 = MaksDiff, 4 = SSIM): 1
Masukkan ambang batas kesalahan:
- Untuk Varians, coba nilai antara 10 dan 50.
- Untuk MAD, coba nilai antara 5 dan 20.
- Untuk MaksDiff, coba nilai antara 20 dan 100.
- Untuk SSIM, gunakan nilai antara 0 dan 1 (misal, 0,90 sampai 0,99).
Input Anda: 25
Masukkan ukuran blok minimum: 4
Masukkan rasio kompresi target (misal, 2,0 untuk kompresi 2:1, 0 untuk menonaktifkan penyesuaian otomatis): 0
Masukkan path absolut untuk output gambar terkompresi: D:\Projects\stima\Tucil2-10123004\test\compressed\hasil1.jpg
Masukkan path absolut untuk output GIF (kosongkan untuk melewati): D:\Projects\stima\Tucil2-10123004\test\compressed\hasil1_gif.gif
```

----- Laporan Kompresi -----
Waktu eksekusi: 3.74772 detik
Ukuran gambar asli: 2764800 byte
Perkiraan ukuran terkompresi: 433048 byte
Persentase kompresi: 84.3371 %
Kedalaman quadtree: 8
Jumlah total node: 20621



Test Case #2

```
Masukkan path absolut gambar yang akan dikompresi: D:\Projects\stima\Tucil2-10123004\test\original\test2.jpg
Masukkan metode pengukuran kesalahan (1 = Varians, 2 = MAD, 3 = MaksDiff, 4 = SSIM): 2
Masukkan ambang batas kesalahan:
- Untuk Varians, coba nilai antara 10 dan 50.
- Untuk MAD, coba nilai antara 5 dan 20.
- Untuk MaksDiff, coba nilai antara 20 dan 100.
- Untuk SSIM, gunakan nilai antara 0 dan 1 (misal, 0,90 sampai 0,99).
Input Anda: 10
Masukkan ukuran blok minimum: 8
Masukkan rasio kompresi target (misal, 2,0 untuk kompresi 2:1, 0 untuk menonaktifkan penyesuaian otomatis): 0
Masukkan path absolut untuk output gambar terkompresi: D:\Projects\stima\Tucil2-10123004\test\compressed\hasil2.jpg
Masukkan path absolut untuk output GIF (kosongkan untuk melewati): D:\Projects\stima\Tucil2-10123004\test\compressed\hasil2.gif
```

```
----- Laporan Kompresi -----
Waktu eksekusi: 3.74772 detik
Ukuran gambar asli: 2764800 byte
Perkiraan ukuran terkompresi: 433048 byte
Persentase kompresi: 84.3371 %
Kedalaman quadtree: 8
Jumlah total node: 20621
```



Test Case #3

```
Masukkan path absolut gambar yang akan dikompresi: D:\Projects\stima\Tucil2-10123004\test\original\test3.jpg
Masukkan metode pengukuran kesalahan (1 = Varians, 2 = MAD, 3 = MaksDiff, 4 = SSIM): 3
Masukkan ambang batas kesalahan:
- Untuk Varians, coba nilai antara 10 dan 50.
- Untuk MAD, coba nilai antara 5 dan 20.
- Untuk MaksDiff, coba nilai antara 20 dan 100.
- Untuk SSIM, gunakan nilai antara 0 dan 1 (misal, 0,90 sampai 0,99).
Input Anda: 50
Masukkan ukuran blok minimum: 2
Masukkan rasio kompresi target (misal, 2,0 untuk kompresi 2:1, 0 untuk menonaktifkan penyesuaian otomatis): 0
Masukkan path absolut untuk output gambar terkompresi: D:\Projects\stima\Tucil2-10123004\test\compressed\hasil3.jpg
Masukkan path absolut untuk output GIF (kosongkan untuk melewati): D:\Projects\stima\Tucil2-10123004\test\compressed\hasil3.gif.gif
```

```
----- Laporan Kompresi -----
Waktu eksekusi: 40.8667 detik
Ukuran gambar asli: 28853760 byte
Perkiraan ukuran terkompresi: 2124472 byte
Persentase kompresi: 92.6371 %
Kedalaman quadtree: 11
Jumlah total node: 101165
```



Test Case #4

```
Masukkan path absolut gambar yang akan dikompresi: D:\Projects\stima\Tucil2-10123004\test\original\test4.jpg  
Masukkan metode pengukuran kesalahan (1 = Varians, 2 = MAD, 3 = MaksDiff, 4 = SSIM): 4
```

```
Masukkan ambang batas kesalahan:
```

- Untuk Varians, coba nilai antara 10 dan 50.
- Untuk MAD, coba nilai antara 5 dan 20.
- Untuk MaksDiff, coba nilai antara 20 dan 100.
- Untuk SSIM, gunakan nilai antara 0 dan 1 (misal, 0,90 sampai 0,99).

```
Input Anda: 0,95
```

```
Masukkan ukuran blok minimum: 5
```

```
Masukkan rasio kompresi target (misal, 2,0 untuk kompresi 2:1, 0 untuk menonaktifkan penyesuaian otomatis): 0
```

```
Masukkan path absolut untuk output gambar terkompresi: D:\Projects\stima\Tucil2-10123004\test\compressed\hasil4.jpg
```

```
Masukkan path absolut untuk output GIF (kosongkan untuk melewati): D:\Projects\stima\Tucil2-10123004\test\compressed\hasil4.gif
```

Waktu eksekusi: 51.8979 detik

Ukuran gambar asli: 28853760 byte

Perkiraan ukuran terkompresi: 7340032 byte

Persentase kompresi: 74.5613 %

Kedalaman quadtree: 9

Jumlah total node: 349525



Test Case #5

```
Masukkan path absolut gambar yang akan dikompresi: D:\Projects\stima\Tucil2-10123004\test\original\test5.jpg
Masukkan metode pengukuran kesalahan (1 = Varians, 2 = MAD, 3 = MaksDiff, 4 = SSIM): 1
Masukkan ambang batas kesalahan:
- Untuk Varians, coba nilai antara 10 dan 50.
- Untuk MAD, coba nilai antara 5 dan 20.
- Untuk MaksDiff, coba nilai antara 20 dan 100.
- Untuk SSIM, gunakan nilai antara 0 dan 1 (misal, 0,90 sampai 0,99).
Input Anda: 20
Masukkan ukuran blok minimum: 8
Masukkan rasio kompresi target (misal, 2.0 untuk kompresi 2:1, 0 untuk menonaktifkan penyesuaian otomatis): 0
Masukkan path absolut untuk output gambar terkompresi: D:\Projects\stima\Tucil2-10123004\test\compressed\hasil5.jpg
Masukkan path absolut untuk output GIF (kosongkan untuk melewati): D:\Projects\stima\Tucil2-10123004\test\compressed\hasil5.gif
```

----- Laporan Kompresi -----

Waktu eksekusi: 42.4117 detik

Ukuran gambar asli: 36067200 byte

Perkiraan ukuran terkompresi: 2348920 byte

Persentase kompresi: 93.4874 %

Kedalaman quadtree: 9

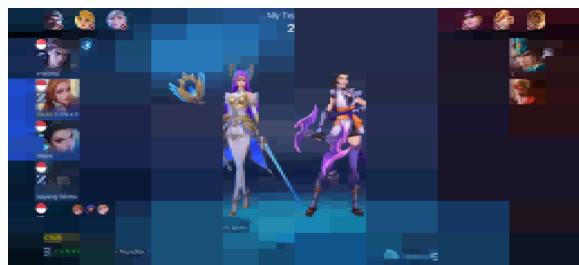
Jumlah total node: 111853



Test Case #6

```
Masukkan path absolut gambar yang akan dikompresi: D:\Projects\stima\Tucil2-10123004\test\original\test6.jpg
Masukkan metode pengukuran kesalahan (1 = Varians, 2 = MAD, 3 = MaksDiff, 4 = SSIM): 2
Masukkan ambang batas kesalahan:
- Untuk Varians, coba nilai antara 10 dan 50.
- Untuk MAD, coba nilai antara 5 dan 20.
- Untuk MaksDiff, coba nilai antara 20 dan 100.
- Untuk SSIM, gunakan nilai antara 0 dan 1 (misal, 0,90 sampai 0,99).
Input Anda: 15
Masukkan ukuran blok minimum: 6
Masukkan rasio kompresi target (misal, 2,0 untuk kompresi 2:1, 0 untuk menonaktifkan penyesuaian otomatis): 0
Masukkan path absolut untuk output gambar terkompresi: D:\Projects\stima\Tucil2-10123004\test\compressed\hasil6.jpg
Masukkan path absolut untuk output GIF (kosongkan untuk melewati): D:\Projects\stima\Tucil2-10123004\test\compressed\hasil6_gif.gif
```

```
----- Laporan Kompresi -----
Waktu eksekusi: 9.18008 detik
Ukuran gambar asli: 9925920 byte
Perkiraan ukuran terkompresi: 233800 byte
Persentase kompresi: 97.6446 %
Kedalaman quadtree: 8
Jumlah total node: 11133
```



Test Case #7

```
Masukkan path absolut gambar yang akan dikompresi: D:\Projects\stima\Tucil2-10123004\test\original\test7.jpg
Masukkan metode pengukuran kesalahan (1 = Varians, 2 = MAD, 3 = MaksDiff, 4 = SSIM): 3
Masukkan ambang batas kesalahan:
- Untuk Varians, coba nilai antara 10 dan 50.
- Untuk MAD, coba nilai antara 5 dan 20.
- Untuk MaksDiff, coba nilai antara 20 dan 100.
- Untuk SSIM, gunakan nilai antara 0 dan 1 (misal, 0,90 sampai 0,99).
Input Anda: 80
Masukkan ukuran blok minimum: 4
Masukkan rasio kompresi target (misal, 2.0 untuk kompresi 2:1, 0 untuk menonaktifkan penyesuaian otomatis): 3.0
Masukkan path absolut untuk output gambar terkompresi: D:\Projects\stima\Tucil2-10123004\test\compressed\hasil7.jpg
Masukkan path absolut untuk output GIF (kosongkan untuk melewati): D:\Projects\stima\Tucil2-10123004\test\compressed\hasil7.gif
Selesai. Klik pada 2020-10-21_22-20-00.html
```

----- Laporan Kompresi -----

Waktu eksekusi: 45.5854 detik

Ukuran gambar asli: 28853760 byte

Perkiraan ukuran terkompresi: 842296 byte

Persentase kompresi: 97.0808 %

Kedalaman quadtree: 10

Jumlah total node: 40109



Analisis Pada Algoritma Kompresi Dengan Quadtree

1. Analisis Algoritma

Algoritma ini menggunakan struktur data Quadtree untuk membagi gambar secara rekursif menjadi 4 kuadran sampai memenuhi salah satu kondisi:

- Error metric (*Varians*, *MAD*, *MaksDiff*, atau *SSIM*) mencapai ambang batas
- Ukuran blok mencapai minimum yang ditentukan
- Rasio kompresi target tercapai (dalam mode penyesuaian otomatis)
-

Fase utama algoritma:

1. Inisialisasi Node: Menghitung warna rata-rata dan error metric untuk setiap blok
2. Pembagian Rekursif: Membagi blok menjadi 4 sub-blok jika belum memenuhi kriteria berhenti
3. Rendering: Membangun gambar terkompresi dari node daun
4. Analisis: Menghitung berbagai metrik kualitas dan kompresi

2. Analisis Kompleksitas Waktu

a. Kompleksitas Per Node (*QuadtreeNode Constructor*)

- Iterasi melalui semua piksel dalam blok: $O(n)$ dimana $n = \text{jumlah piksel dalam blok}$
- Operasi per piksel: 3 penjumlahan + 3 operasi absolut + 3 operasi kuadrat
- Total per node: $O(n)$

b. Pembangunan *Quadtree* (*build_tree*)

Worst case (gambar sangat heterogen):

- Setiap node akan terus membagi sampai ukuran minimum
- Kedalaman maksimum: $\log(\max(\text{width}, \text{height}))$
- Jumlah node total: $O(N)$ dimana $N = \text{total piksel}$
- Total waktu: $O(N \log N)$ karena setiap level memproses $O(N)$ piksel

Best case (gambar homogen):

- Hanya root node yang diproses
- Total waktu: $O(N)$

c. Operasi Lainnya

- Render gambar: $O(M)$ dimana $M = \text{jumlah node daun}$
- Analisis kedalaman: $O(D)$ dimana $D = \text{kedalaman maksimum}$
- Pembuatan GIF: $O(D*N)$ karena perlu render di setiap kedalaman

3. Analisis Kompleksitas Ruang

a. Penyimpanan Quadtree

Setiap node menyimpan:

- 4 pointer anak
- Metadata (box, warna, error)

Ruang per node: $O(1)$

Total ruang: $O(N)$ untuk worst case, $O(M)$ untuk average case ($M < N$)

- b. Penyimpanan Gambar
- Original data: $O(N)$
 - Temporary buffers: $O(N)$ untuk operasi rendering

4. Faktor yang Mempengaruhi Kinerja

Faktor	Pengaruh Pada Kompleksitas
Ukuran Blok Minimum	Batas kedalaman maksimum
Threshold Error	Mengatur jumlah split
Metode Error	Kualitas kompresi dan kompleksitas perhitungan
Homogenitas Gambar Asli	Mengatur jumlah split
Rasio Kompresi Target	Menambah iterasi untuk penyesuaian

5. Penurunan Kompleksitas Waktu

Misalkan gambar berukuran $W \times H$ piksel:

1. Base Case (tidak ada split): $T(n) = O(W * H)$
2. Recursive Case (split): $T(n) = 4T(n/4) + cn$

Dengan Teorema Master diperoleh: $a = 4$, $b = 4$, $c = 1$

Karena $4 = 4^1$, artinya $a = b^d$, sehingga berdasarkan Teorema Master diperoleh kompleksitas: $O(n^1 \log n) = O(n \log n)$

6. Optimasi Potensial

- *Parallel Processing*: Memproses sub-blok secara paralel
- *Early Termination*: Menggunakan heuristik untuk menghentikan split lebih awal
- *Adaptive Block Size*: Ukuran blok minimum yang bervariasi berdasarkan area gambar
- *Memoization*: Menyimpan hasil perhitungan error metric untuk area yang sama

Implementasi Bonus

Membuat GIF

Pembuatan GIF dilakukan pada blok kode berikut:

```
// Buat GIF yang menunjukkan proses kompresi
void create_gif(const string &filename, int delay_ms = 100) {
    GifWriter g;
    GifBegin(&g, filename.c_str(), width, height, delay_ms / 10);
    for (int d = 0; d <= max_depth; d++) {
        unsigned char* frame = create_image_from_depth(d);
        uint8_t* rgba = new uint8_t[width * height * 4];
        for (int i = 0; i < width * height; i++) {
            rgba[i * 4 + 0] = frame[i * 3 + 0];
            rgba[i * 4 + 1] = frame[i * 3 + 1];
            rgba[i * 4 + 2] = frame[i * 3 + 2];
            rgba[i * 4 + 3] = 255;
        }
        GifWriteFrame(&g, rgba, width, height, delay_ms / 10);
        delete[] rgba;
        delete[] frame;
    }
    unsigned char* final_frame =
    create_image_from_depth(max_depth);
    uint8_t* final_rgba = new uint8_t[width * height * 4];
    for (int i = 0; i < width * height; i++) {
        final_rgba[i * 4 + 0] = final_frame[i * 3 + 0];
        final_rgba[i * 4 + 1] = final_frame[i * 3 + 1];
        final_rgba[i * 4 + 2] = final_frame[i * 3 + 2];
        final_rgba[i * 4 + 3] = 255;
    }
    for (int i = 0; i < 4; i++)
        GifWriteFrame(&g, final_rgba, width, height, delay_ms /
10);
    GifEnd(&g);
    delete[] final_rgba;
    delete[] final_frame;
    cout << "GIF telah dibuat di " << filename << "\n";
}
```

Fungsi pembuatan GIF dapat dipanggil ketika sedang memasukkan parameter, jika ingin dipakai maka dapat dimasukkan absolute path untuk GIF hasil

GIF untuk hasil test case dapat diakses pada pranala: [compressed](#)

Target Presentase Kompresi

Penyesuaian target presentase kompresi dilakukan pada blok kode ini dengan *Binary Search*

```
// Penyesuaian otomatis ambang batas (hanya untuk metode Varians)
if (config.target_compression > 0 && config.error_method == 1) {
    float low = 0.0f, high = 255.0f;
    const int iterations = 15;
    for (int i = 0; i < iterations; i++) {
        config.error_threshold = (low + high) / 2.0f;
        Quadtree tempTree(img_data, width, height, channels,
config);
        int leafCount =
tempTree.get_leaf_nodes(tempTree.get_max_depth()).size();
        int origSize = width * height * 3;
        int estSize = leafCount * (sizeof(RGB) + sizeof(Box));
        double achievedRatio = static_cast<double>(origSize) /
estSize;
        if (achievedRatio < config.target_compression)
            low = config.error_threshold;
        else
            high = config.error_threshold;
    }
    cout << "Ambang batas Varians disesuaikan otomatis menjadi: "
<< config.error_threshold << "\n";
    tree = Quadtree(img_data, width, height, channels, config);
}
```

Lampiran

Github: https://github.com/adielrum/Tucil2_10123004.git