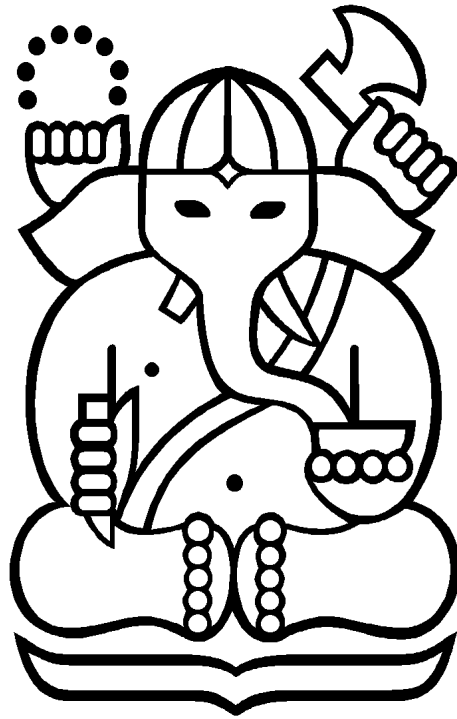


Laporan Tugas Kecil 2 Strategi Algoritma
IF-2211 2024/2025 Kompresi Gambar dengan Metode Quadtree



Disusun oleh :

Noumisyifa Nabila Nareswari – 13523058 – K01

Diyah Susan Nugrahani – 13523080 – K02

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2025

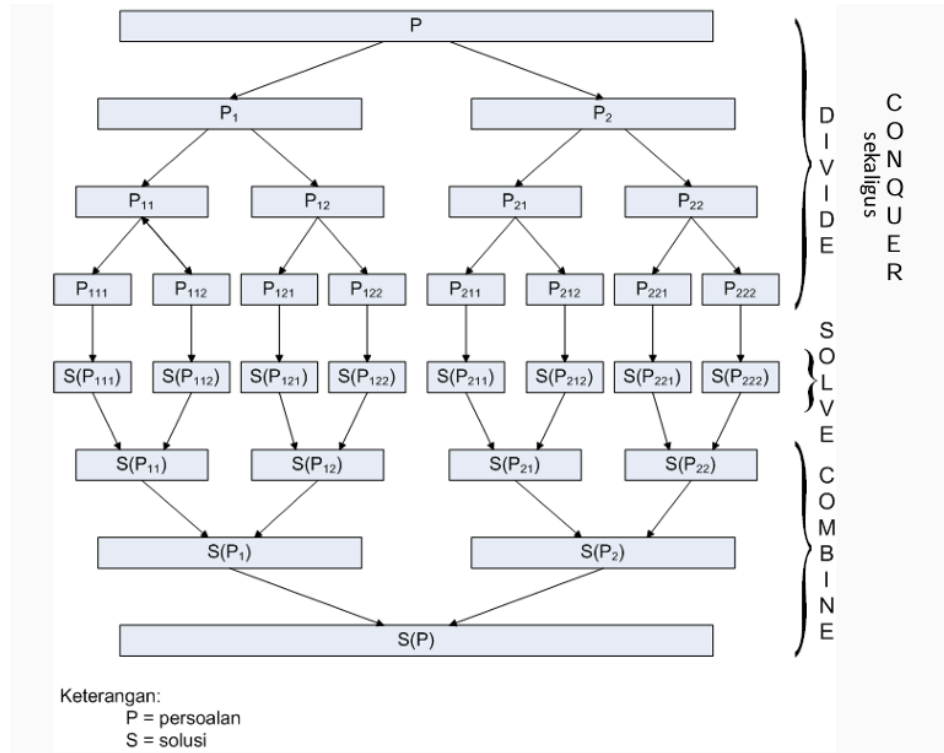
DAFTAR ISI

DAFTAR ISI	1
A. Algoritma Divide and Conquer	2
1. Definisi Divide and Conquer	2
2. Quadtree	2
3. Penjelasan Algoritma Quadtree dalam Kompresi Gambar	3
B. Source Program	7
1. Main.cpp	7
2. RGB.hpp	9
3. QuadTree.cpp	9
4. QuadTreeNode.cpp	11
C. Tangkapan layar input dan output	15
D. Hasil analisis percobaan algoritma divide and conquer	27
1. Kompleksitas algoritma	27
2. Analisis hasil uji	29
E. Penjelasan implementasi bonus	30
1. Penjelasan bonus Structural Similarity Index (SSIM)	30
2. Penjelasan bonus target persentase kompresi	31
F. Lampiran	34

A. Algoritma *Divide and Conquer*

1. Definisi *Divide and Conquer*

Divide and conquer merupakan sebuah algoritma yang menggunakan konsep *divide*, *conquer*, and *combine*. *Divide* membagi persoalan menjadi beberapa sub-persoalan yang memiliki kemiripan dengan persoalan semula namun memiliki ukuran yang lebih kecil. *Conquer*, menyesuaikan solusi masing-masing sub-persoalan secara langsung jika ukurannya kecil atau secara rekursif jika ukurannya masih terlalu besar. Kemudian solusi-solusi tersebut akan di-*combine* sehingga membentuk solusi untuk persoalan semula.

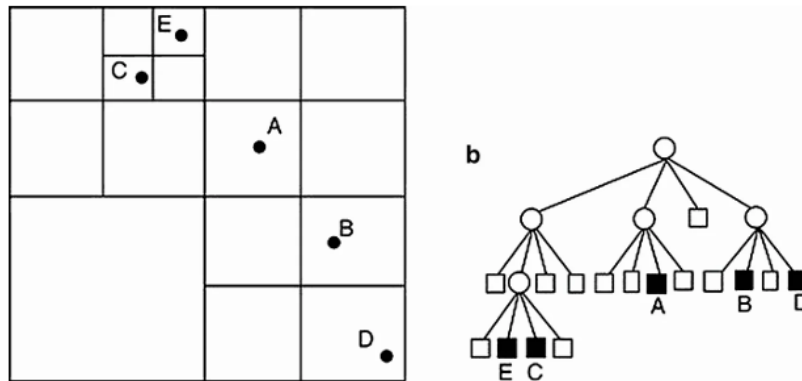


Gambar 1 Ilustrasi *Divide and Conquer* (sumber :

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf))

2. Quadtree

Quadtree merupakan sebuah struktur data yang digunakan untuk membagi data menjadi bagian yang lebih kecil. *Quadtree* memiliki struktur di mana setiap node asli akan memiliki maksimal empat node anak. QuadTree bekerja dengan prinsip divide and conquer pada ruang dua dimensi. Proses pembuatan QuadTree dimulai dengan satu node akar yang kemudian node tersebut dibagi menjadi empat bagian.



Gambar 2. Struktur Data Quadtree dalam Kompresi Gambar (Sumber : <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>)

3. Penjelasan Algoritma Quadtree dalam Kompresi Gambar

Konsep pemanfaatan *quadtree* untuk kompresi gambar adalah dengan membagi gambar yang berukuran besar menjadi empat bagian dan mengevaluasi bagian tersebut dengan membandingkan komposisi warna *red*, *green*, and *blue* (*RGB*) pada piksel-piksel dalam gambar tersebut. Jika bagian tersebut tidak seragam maka bagian tersebut akan terus dibagi hingga mencapai situasi seragam di bagian tersebut atau telah mencapai ukuran minimum yang ditentukan. Penggunaan *quadtree* ini dapat mengurangi ukuran file gambar tanpa mengorbankan detail penting pada gambar tersebut.

Dalam menentukan apakah suatu bagian dapat dikatakan seragam didasarkan pada aspek metode perhitungan variansi, *threshold* variansi, dan *minimum block size*. Nilai hasil perhitungan variansi akan dibandingkan dengan *threshold*. Jika variansi di atas *threshold*, ukuran blok masih lebih besar dari *minimum block size*, dan ukuran blok setelah dibagi empat tidak kurang dari *minimum block size* maka blok tersebut akan dibagi menjadi empat dan proses akan dilanjutkan untuk setiap sub-bagian. Jika salah satu kondisi tersebut tidak terpenuhi maka proses pembagian dihentikan dan bagian tersebut akan menjadi daun. Untuk blok daun akan dilakukan normalisasi warna blok sesuai dengan rata-rata nilai *RGB* blok tersebut.

Program ini mengimplementasikan konsep QuadTree dengan kelas utama yaitu QuadTree dan struktur data QuadTreeNode. QuadTree dalam hal ini merupakan kelas utama yang mengelola struktur pohon secara keseluruhan, sedangkan QuadTreeNode adalah struktur yang merepresentasikan setiap node dalam pohon. Atribut yang terdapat pada struktur data QuadTreeNode adalah sebagai berikut :

```
struct QuadTreeNode {
    QuadTreeNode* topLeft;
    QuadTreeNode* topRight;
    QuadTreeNode* bottomLeft;
    QuadTreeNode* bottomRight;
};
```

```

int x, y, width, height;
bool isLeaf;
RGB meanColor;
};

```

Di dalam QuadTreeNode terdapat method buildNode() dan buildNodeSSIM() yang digunakan untuk membuat node pada suatu pohon. Method ini merupakan inti dari algoritma divide and conquer, dimana dalam method ditentukan apakah sebuah bagian pada suatu block gambar cukup seragam atau tidak. Alur dari algoritma divide and conquer dalam pembentukan node pada QuadTree adalah sebagai berikut :

- a. Cek apakah node kosong, jika kosong keluar program dan jika tidak kosong lanjut ke tahap berikutnya
- b. Set properti node dengan value yang telah di passing di pemanggilan method buildNode
- c. Update maximum depth jika current depth nya lebih besar dari maximum depth saat ini
- d. Periksa apakah blok saat ini sudah seragam atau belum, jika sudah seragam maka block tersebut akan menjadi leaf dan proses untuk blok tersebut selesai. Namun jika belum seragam, maka blok tersebut akan dibagi menjadi empat bagian.
- e. Empat sub-bagian yang ada pada node tersebut akan diproses secara rekursif dengan memanggil fungsi buildNode lagi sampai didapatkan keadaan node sudah seragam dan tidak perlu dibagi lagi.
- f. Update node count setiap kali memanggil buildNode

ALGORITMA

```

Procedure buildNode (node, image, x, y, width, height, threshold,
currentDepth, maxDepth, nodeCount, errorMethod, minBlockSize)
    if ( node == NULL) then
        → {keluar program karena node kosong}
    end if

    {set properti node}
    node.x ← x
    node.y ← y
    node.width ← width
    node.height ← height

    {update maximum depth}
    if (currentDepth > maxDepth) then
        maxDepth ← currentDepth
    end if

    {cek apakah seragam}
    isUniform ← errorValidation(errorMethod, image, x, y,
width, height, threshold, node.meanColor)

    if isUniform or width ≤ minBlockSize or

```

```

height ≤ minBlockSize then
    node.isLeaf ← TRUE
    node.topLeft ← NULL
    node.topRight ← NULL
    node.bottomLeft ← NULL
    node.bottomRight ← NULL
    nodeCount ← nodeCount + 1
    → {keluar program}
end if

{kasus tidak seragam sehingga harus dibagi empat sub-bagian}
node.isLeaf ← false

halfWidth ← width div 2
halfHeight ← height div 2
rightWidth ← width - halfWidth
bottomHeight ← height - halfHeight

{proses top-left quadrant}
node.topLeft ← NEW QuadTreeNode()
buildNode(node.topLeft, image, x, y, halfWidth, halfHeight,
threshold, currentDepth+1, maxDepth, nodeCount, errorMethod,
minBlockSize)

{proses top-right quadrant}
node.topRight ← NEW QuadTreeNode()
buildNode(node.topRight, image, x+halfWidth, y, rightWidth,
halfHeight, threshold, currentDepth+1, maxDepth, nodeCount,
errorMethod, minBlockSize)

{proses bottom-left quadrant}
node.bottomLeft ← NEW QuadTreeNode()
buildNode(node.bottomLeft, image, x, y+halfHeight, halfWidth,
bottomHeight, threshold, currentDepth+1, maxDepth,
nodeCount, errorMethod, minBlockSize)

{proses bottom-right quadrant}
node.bottomRight ← NEW QuadTreeNode()
buildNode(node.bottomRight, image, x+halfWidth,
y+halfHeight, rightWidth, bottomHeight, threshold,
currentDepth+1, maxDepth, nodeCount, errorMethod,
minBlockSize)

nodeCount ← nodeCount + 1

```

{untuk method buildNodeSSIM memiliki ide algoritma yang sama namun yang membedakan adalah parameternya}

Untuk mengecek keseragaman suatu block gambar digunakan beberapa pendekatan *error measurement methods*. Beberapa diantaranya adalah metode *variance*, *mean absolute deviation (MAD)*, *max pixel difference (MPD)*, *entropy*, dan structural similarity index *SSIM* (bonus).

Metode	Formula
Variance	$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	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
	N = Banyaknya 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
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 = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i 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 = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok
	$P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)
[Bonus] Structural Similarity Index (SSIM) (Referensi tambahan)	$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$
	$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$
	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.

Gambar 3. Formula Metode Pengukuran Error (sumber : Spesifikasi Tugas Kecil 2 IF2211 Strategi Algoritma)

Method buildNode yang telah dibuat akan dipanggil pada class QuadTree karena pada kelas ini lah pohon utama yang berisikan node-node pohon dibuat.

ALGORITMA

```
Procedure buildTree(image, threshold, errorMethod, minBlockSize)
    root ← new QuadTreeNode()
    maxDepth ← 0
    nodeCount ← 0
    root.buildNode(root, image, 0, 0, image[0].size(),
        image.size(), threshold, 1, maxDepth, nodeCount,
        errorMethod, minBlockSize)
```

Setelah procedure build Tree dijalankan maka akan terbentuk image hasil modifikasi setelah melalui proses divide and conquer dengan QuadTree. Kemudian gambar tersebut akan direkonstruksi dan disimpan menjadi output gambar hasil kompresi.

B. Source Program

1. Main.cpp

Adalah program utama yang dijalankan dan berisikan penggunaan dari berbagai fungsi dan prosedur yang telah dibuat.

```
#include <bits/stdc++.h>
using namespace std;
#include "Utils.hpp"
#include "QuadTree.hpp"
#include <chrono>

int main(){
    string inputImagePath, errorMethod, outputImagePath;
    vector<vector<RGB>> image;
    float threshold, targetCompression, bestThreshold;
    int minBlockSize, maxDepth, nodeCount;

    // Handle input and process the image
    inputHandler(inputImagePath, image, errorMethod, threshold,
minBlockSize, targetCompression, outputImagePath);

    // Start menghitung waktu eksekusi
    auto start = chrono::high_resolution_clock::now();

    QuadTree qt;
```



```

//Implementasi rekursi di sini
if(targetCompression == 0){
    vector<vector<RGB>> imageOri = image;
    if (errorMethod == "ssim"){
        qt.buildTree(image, imageOri, threshold, errorMethod,
minBlockSize);
    }
    else{
        qt.buildTree(image, threshold, errorMethod, minBlockSize);
    }
    maxDepth = qt.maxDepth;
    nodeCount = qt.nodeCount;
    qt.reconstructImage(image);
    saveCompressedImage(image, outputImagePath);
} else {
    if(errorMethod == "ssim"){
        vector<vector<RGB>> imageOri = image;
        bestThreshold = ssimPercentageCompression(image, imageOri,
inputImagePath, outputImagePath, errorMethod, minBlockSize,
targetCompression, maxDepth, nodeCount);
        qt.buildTree(image, imageOri, bestThreshold, errorMethod,
minBlockSize);
    } else{
        bestThreshold = standardPercentageCompression(image,
inputImagePath, outputImagePath, errorMethod, minBlockSize,
targetCompression, maxDepth, nodeCount);
        qt.buildTree(image, bestThreshold, errorMethod,
minBlockSize);
    }
}

// Rekursi selesai
auto end = chrono::high_resolution_clock::now();
auto duration = chrono::duration_cast<chrono::milliseconds>(end -
start);

    outputHandler(outputImagePath, inputImagePath, maxDepth,
nodeCount, duration);

```

```
    return 0;
}
```

2. RGB.hpp

ADT sederhana untuk membangun tipe RGB. Berisikan atribut r representasi dari warna merah, g representasi dari warna hijau, dan b representasi dari warna biru. Dalam tugas ini nilai alpha tidak dipertimbangkan dalam kalkulasi.

```
#ifndef RGB_HPP
#define RGB_HPP

#include <cstdint>

struct RGB {
    uint8_t r;
    uint8_t g;
    uint8_t b;
};

#endif
```

3. QuadTree.cpp

Merupakan sebuah ADT dengan spesifikasi tambahan yang secara pasti membuat empat simpul untuk tiap satu simpul yang bercabang. Fungsi utama dari ADT ini adalah untuk menampung data kedalaman, jumlah simpul, menginisiasi fungsi rekursi untuk membangun tree dari sekumpulan simpul, yaitu prosedur buildNode(), dan menginisiasi fungsi rekursi untuk merekonstruksi ulang vector of RGB menjadi gambar, yaitu prosedur reconstructRecursive(). Fungsi utama tadi diimplementasikan dalam prosedur buildTree() dan reconstructImage(), metode lain yang ada hanyalah metode default yang harus ada seperti konstruktor dan destruktur.

```
#include "QuadTree.hpp"
#include "ErrorMeasures.hpp"

QuadTree::QuadTree() : root(nullptr), maxDepth(0), nodeCount(0) {}

QuadTree::~QuadTree() {
```

```

        destroyTree(root);
    }

void QuadTree::destroyTree(QuadTreeNode* node) {
    if (node == nullptr) return;

    // rekursif hapus semua anak
    destroyTree(node->topLeft);
    destroyTree(node->topRight);
    destroyTree(node->bottomLeft);
    destroyTree(node->bottomRight);

    delete node;
}

// intinya ini bakal manggil buildNode, dipisah supaya lebih modular
void QuadTree::buildTree(const std::vector<std::vector<RGB>>& image,
float threshold, const std::string& errorMethod, int minBlockSize) {
    root = new QuadTreeNode();
    maxDepth = 0;
    nodeCount = 0;

    root->buildNode(root, image, 0, 0, image[0].size(), image.size(),
threshold, 1, maxDepth, nodeCount, errorMethod, minBlockSize);
}

// yang ini untuk kasus SSIM
void QuadTree::buildTree(const std::vector<std::vector<RGB>>& image1,
const std::vector<std::vector<RGB>>& image2, float threshold, const
std::string& errorMethod, int minBlockSize) {
    root = new QuadTreeNode();
    maxDepth = 0;
    nodeCount = 0;

    root->buildNodeSSIM(root, image1, image2, 0, 0, image1[0].size(),
image1.size(), threshold, 1, maxDepth, nodeCount, minBlockSize);
}

// ini baru, buat simpan gambarnya supaya ngga besar
void QuadTree::reconstructImage(std::vector<std::vector<RGB>>& image)

```

```

{
    if (!root) return;
    reconstructRecursive(root, image);
}

void QuadTree::reconstructRecursive(QuadTreeNode* node,
std::vector<std::vector<RGB>>& image) {
    if (node->isLeaf) {
        // isi bagian dengan rata-rata warna
        for (int y = node->y; y < node->y + node->height; ++y) {
            for (int x = node->x; x < node->x + node->width; ++x) {
                image[y][x] = node->meanColor;
            }
        }
    } else {
        reconstructRecursive(node->topLeft, image);
        reconstructRecursive(node->topRight, image);
        reconstructRecursive(node->bottomLeft, image);
        reconstructRecursive(node->bottomRight, image);
    }
}
}

```

4. QuadTreeNode.cpp

Merupakan sebuah ADT yang merepresentasikan simpul dari ADT QuadTree. Fungsi utama dari ADT ini adalah untuk menjalankan rekursi untuk membentuk QuadTree dengan aturan seperti yang telah dijelaskan pada bagian sebelumnya. Fungsi utama tadi diimplementasikan dalam prosedur buildNode() dan buildNodeSSIM(), metode lain yang ada hanyalah metode default yang harus ada seperti konstruktor.

```

#include "QuadTreeNode.hpp"
#include <iostream>
#include <vector>
#include <cmath>

QuadTreeNode::QuadTreeNode()
    : topLeft(nullptr), topRight(nullptr), bottomLeft(nullptr),

```

```

bottomRight(nullptr),
    x(0), y(0), width(0), height(0), isLeaf(true), meanColor{0, 0,
0} {}

void QuadTreeNode::buildNode(QuadTreeNode*& node, const
std::vector<std::vector<RGB>>& image,
    int x, int y, int width, int height, float threshold,
    int currentDepth, int& maxDepth, int& nodeCount, const
std::string& errorMethod, int minBlockSize)
{
    // membangun node pada quadtree, objektifnya adalah jika blok
    masih belum seragam maka blok akan dibagi menjadi empat
    // hal tersebut dilakukan terus menerus secara rekursif hingga blok
    tersebut seragam atau ukurannya tidak kurang dari minBlockSize
    if (!node) return;

    //inisialisasi
    node->x = x;
    node->y = y;
    node->width = width;
    node->height = height;

    // update maxDepth
    if (currentDepth > maxDepth) {
        maxDepth = currentDepth;
    }

    bool isUniform = errorValidation(errorMethod, image, x, y, width,
height, threshold, node->meanColor);

    if (isUniform || width <= minBlockSize || height <= minBlockSize){
        node->isLeaf = true;
        node->topLeft = nullptr;
        node->topRight = nullptr;
        node->bottomLeft = nullptr;
        node->bottomRight = nullptr;
        nodeCount ++;
        return;
    }
}

```

```

    // not uniform, split
    node->isLeaf = false;
    int halfWidth = width / 2;
    int halfHeight = height / 2;
    int rightWidth = width - halfWidth;
    int bottomHeight = height - halfHeight;

    node->topLeft = new QuadTreeNode();
    buildNode(node->topLeft, image, x, y, halfWidth, halfHeight,
threshold, currentDepth + 1, maxDepth, nodeCount, errorMethod,
minBlockSize);

    node->topRight = new QuadTreeNode();
    buildNode(node->topRight, image, x + halfWidth, y, rightWidth,
halfHeight, threshold, currentDepth + 1, maxDepth, nodeCount,
errorMethod, minBlockSize);

    node->bottomLeft = new QuadTreeNode();
    buildNode(node->bottomLeft, image, x, y + halfHeight, halfWidth,
bottomHeight, threshold, currentDepth + 1, maxDepth, nodeCount,
errorMethod, minBlockSize);

    node->bottomRight = new QuadTreeNode();
    buildNode(node->bottomRight, image, x + halfWidth, y + halfHeight,
rightWidth, bottomHeight, threshold, currentDepth + 1, maxDepth,
nodeCount, errorMethod, minBlockSize);

    nodeCount ++;
}

void QuadTreeNode::buildNodeSSIM(QuadTreeNode*& node, const
std::vector<std::vector<RGB>>& image1, const
std::vector<std::vector<RGB>>& image2,
    int x, int y, int width, int height, float threshold, int
currentDepth, int& maxDepth, int& nodeCount, int minBlockSize)
{
    // membangun node pada quadtree, objektifnya adalah jika blok
    masih belum seragam maka blok akan dibagi menjadi empat

```

```

    // hal tersebut dilakukan terus menerus secara rekursif hingg blok
    tersebut seragam atau ukurannya tidak kurang dari minBlockSize
    // untuk SSIM ini akan comparing blok ori dengan blok hasil
    kompresi
    if (!node) return;

    node->x = x;
    node->y = y;
    node->width = width;
    node->height = height;

    // update maxDepth
    if (currentDepth > maxDepth) {
        maxDepth = currentDepth;
    }

    bool isUniform = ssim(image1, x, y, width, height, threshold,
node->meanColor);

    if (isUniform || width <= minBlockSize || height <= minBlockSize){
        node->isLeaf = true;
        node->topLeft = nullptr;
        node->topRight = nullptr;
        node->bottomLeft = nullptr;
        node->bottomRight = nullptr;
        nodeCount ++;
        return;
    }
    // jika nggak uniform, split node jadi empat bagian
    node->isLeaf = false;
    int halfWidth = width / 2;
    int halfHeight = height / 2;
    int rightWidth = width - halfWidth;
    int bottomHeight = height - halfHeight;

    // rekursif membuat empat anak
    node->topLeft = new QuadTreeNode();
    buildNodeSSIM(node->topLeft, image1, image2, x, y, halfWidth,
halfHeight, threshold, currentDepth + 1, maxDepth, nodeCount,

```

```

minBlockSize);

    node->topRight = new QuadTreeNode();
    buildNodeSSIM(node->topRight, image1, image2, x + halfWidth, y,
rightWidth, halfHeight, threshold, currentDepth + 1, maxDepth,
nodeCount, minBlockSize);

    node->bottomLeft = new QuadTreeNode();
    buildNodeSSIM(node->bottomLeft, image1, image2, x, y + halfHeight,
halfWidth, bottomHeight, threshold, currentDepth + 1, maxDepth,
nodeCount, minBlockSize);

    node->bottomRight = new QuadTreeNode();
    buildNodeSSIM(node->bottomRight, image1, image2, x + halfWidth, y
+ halfHeight, rightWidth, bottomHeight, threshold, currentDepth + 1,
maxDepth, nodeCount, minBlockSize);

    nodeCount ++;
}

```


Sebenarnya terdapat satu file lagi yaitu Utils.cpp, namun file ini hanya berisikan hal-hal teknis seperti validasi input, memproses gambar menjadi vector 2d *of* RGB, dan penampilan output yang tentunya akan sangat panjang dan tidak ada hubungannya dengan inti algoritma *divide and conquer* yang menjadi fokus utama di tugas kali ini.


C. Tangkapan layar *input* dan *output*

Berikut adalah tangkapan layar hasil uji program.


HASIL UJI 1 (VARIANCE)

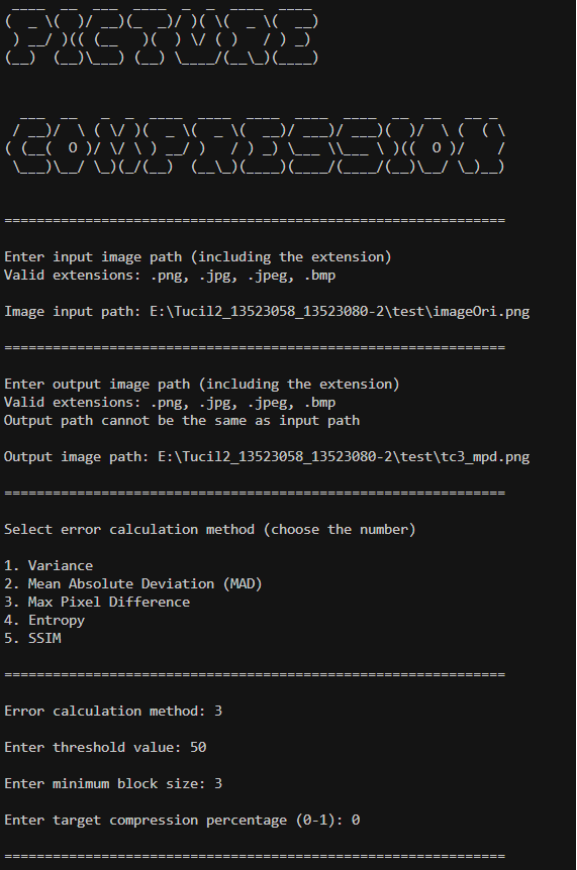
<p><i>Input Gambar</i></p>	
<p><i>Input non gambar</i></p>	<pre> (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) ===== Enter input image path (including the extension) Valid extensions: .png, .jpg, .jpeg, .bmp Image input path: E:\Tucil2_13523058_13523080-2\test\imageOri.png ===== Enter output image path (including the extension) Valid extensions: .png, .jpg, .jpeg, .bmp Output path cannot be the same as input path Output image path: E:\Tucil2_13523058_13523080-2\test\tc1_variance.png ===== Select error calculation method (choose the number) 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. SSIM ===== Error calculation method: 1 Enter threshold value: 400 Enter minimum block size: 3 Enter target compression percentage (0-1): 0 ===== </pre>

<i>Output Gambar</i>	
<i>Output non gambar</i>	<pre> ===== Output image successfully rendered to: E:\Tuc112_13523058_13523080-2\test\tcl_variance.png Compression execution duration: 217 ms Input image size: 334 KB Output image size: 26 KB Compression ratio: 92.2156% reduction Max depth of quadtree: 9 Total nodes in quadtree: 2417 ===== </pre>

HASIL Uji 2 (MAD)	
<i>Input Gambar</i>	

<i>Output non gambar</i>	<pre> ===== Output image successfully rendered to: E:\Tuci12_13523058_13523080-2\test\tc2_mad.png Compression execution duration: 107 ms Input image size: 334 KB Output image size: 20 KB Compression ratio: 94.012% reduction Max depth of quadtree: 9 Total nodes in quadtree: 1005 ===== </pre>
--------------------------	---

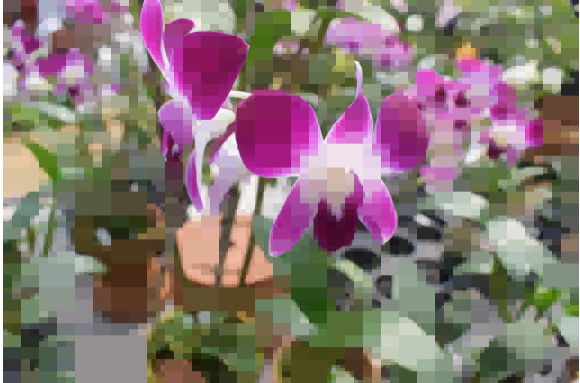
HASIL UJI 3 (MPD)	
<i>Input Gambar</i>	


<p><i>Input non gambar</i></p>	 <pre> (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) (_) ===== Enter input image path (including the extension) Valid extensions: .png, .jpg, .jpeg, .bmp Image input path: E:\Tucil2_13523058_13523080-2\test\imageOri.png ===== Enter output image path (including the extension) Valid extensions: .png, .jpg, .jpeg, .bmp Output path cannot be the same as input path Output image path: E:\Tucil2_13523058_13523080-2\test\tc3_mpd.png ===== Select error calculation method (choose the number) 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. SSIM ===== Error calculation method: 3 Enter threshold value: 50 Enter minimum block size: 3 Enter target compression percentage (0-1): 0 ===== </pre>
<p><i>Output Gambar</i></p>	

<i>Output non gambar</i>	<pre> ===== Output image successfully rendered to: E:\Tucil2_13523080-2\test\tc3_mpd.png Compression execution duration: 589 ms Input image size: 334 KB Output image size: 41 KB Compression ratio: 87.7246% reduction Max depth of quadtree: 9 Total nodes in quadtree: 6349 ===== </pre>
--------------------------	---

HASIL UJI 4 (entropy)	
<i>Input Gambar</i>	

HASIL UJI 5 (Big size MB)	
Input Gambar	
Input non gambar	<pre> { _ \ () / _ \ () / _ \ () / _ \ () / _ \ () / _ \ () / } _ } (((_ \ () / _ \ () / _ \ () / _ \ () / _ \ () / { _ \ () / _ \ () / _ \ () / _ \ () / _ \ () / _ \ () / / _ \ () / _ \ () / _ \ () / _ \ () / _ \ () / _ \ () / (((0) / _ \ () / _ \ () / _ \ () / _ \ () / _ \ () / \ _ \ () / _ \ () / _ \ () / _ \ () / _ \ () / _ \ () / ===== Enter input image path (including the extension) Valid extensions: .png, .jpg, .jpeg, .bmp Image input path: E:\Tucil2_13523058_13523080-2\test\anggrekOri.jpg ===== Enter output image path (including the extension) Valid extensions: .png, .jpg, .jpeg, .bmp Output path cannot be the same as input path Output image path: E:\Tucil2_13523058_13523080-2\test\tc5_bigfile.jpg ===== Select error calculation method (choose the number) 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. SSIM ===== Error calculation method: 1 Enter threshold value: 500 Enter minimum block size: 3 Enter target compression percentage (0-1): 0 </pre>

<i>Output Gambar</i>	
<i>Output non gambar</i>	<pre> ===== Output image successfully rendered to: E:\Tuc12_13523058_13523080-2\test\tc5_bigfile.jpg Compression execution duration: 11799 ms Input image size: 5881 KB Output image size: 773 KB Compression ratio: 86.856% reduction Max depth of quadtree: 12 Total nodes in quadtree: 19585 ===== </pre>

HASIL UJI 6 (SSIM)	
<i>Input Gambar</i>	

<i>Output non gambar</i>	<pre> ===== Output image successfully rendered to: E:\Lucil2_13523058_13523080-2\test\tc6_ssim.png Compression execution duration: 194 ms Input image size: 334 KB Output image size: 76 KB Compression ratio: 77.2455% reduction Max depth of quadtree: 9 Total nodes in quadtree: 18509 ===== </pre>
--------------------------	--

HASIL UJI 7	
<i>Input Gambar</i>	

<p><i>Input non gambar</i></p>	<pre> { } { } { } { } { } { } { } { } } { } { } { } { } { } { } { } { } { } { } { } { } { } { } { } { } ===== Enter input image path (including the extension) Valid extensions: .png, .jpg, .jpeg, .bmp Image input path: E:\Tucil2_13523058_13523080-2\test\imageOri.png ===== Enter output image path (including the extension) Valid extensions: .png, .jpg, .jpeg, .bmp Output path cannot be the same as input path Output image path: E:\Tucil2_13523058_13523080-2\test\tc7_percentage.png ===== Select error calculation method (choose the number) 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. SSIM ===== Error calculation method: 4 Enter threshold value: 4 Enter minimum block size: 4 Enter target compression percentage (0-1): 0.8 ===== </pre>
 <p><i>Output Gambar</i></p>	
<p><i>Output non gambar</i></p>	<pre> ===== Output image successfully rendered to: E:\Tucil2_13523058_13523080-2\test\tc7_percentage.png Compression execution duration: 1054 ms Input image size: 334 KB Output image size: 65 KB Compression ratio: 80.5389% reduction Max depth of quadtree: 9 Total nodes in quadtree: 13885 ===== </pre>

D. Hasil analisis percobaan algoritma *divide and conquer*

1. Kompleksitas algoritma

Didefinisikan $T(n)$ sebagai waktu untuk memproses area gambar $n \times n$ dengan

$$n = \max(a, b)$$

Dimana a mewakili panjang pixel gambar sedangkan b mewakili lebar pixel gambar. Di dalam fungsi `buildNode` dipanggil fungsi `errorValidation()` yang memiliki kompleksitas

waktu $O(n^2)$ karena di dalamnya terdapat nested loop untuk menghitung nilai RGB yang ada dalam tiap pixel dengan salah satu contoh seperti berikut.

```
for (int i = y; i < y + height; ++i) {
    for (int j = x; j < x + width; ++j) {
        // update min values
        minR = min(minR, image[i][j].r);
        minG = min(minG, image[i][j].g);
        minB = min(minB, image[i][j].b);

        // update max values
        maxR = max(maxR, image[i][j].r);
        maxG = max(maxG, image[i][j].g);
        maxB = max(maxB, image[i][j].b);
    }
}
```

Selanjutnya dipanggil empat rekursif untuk bagian dengan ukuran $n/2 \times n/2$. Dengan begitu dapat disimpulkan bahwa kompleksitas waktu $T(n)$ sebagai berikut:

$$T(n) = 4T(n/2) + \Theta(n^2)$$

Melihat bentuk dari persamaan $T(n)$ yang dihasilkan, kompleksitas waktu dapat dianalisis lebih lanjut menggunakan teorema master dengan nilai $a = 4$, $b = 2$, dan $f(n) = \Theta(n^2) = n^{(a \log b)}$. Dengan menggunakan teorema master kasus 2 ($a = b^d$), maka dapat disimpulkan bahwa

$$T(n) = O(n^2 \log n)$$

Kasus terbaik adalah kasus ketika semua blok langsung seragam sehingga tidak ada pemanggilan fungsi rekursi sehingga yang dipanggil hanyalah `errorValidation()` sebanyak satu kali. Dapat disimpulkan bahwa untuk kasus terbaik, kompleksitas waktu adalah sebagai berikut:

$$O(n^2)$$

Sedangkan kompleksitas waktu terburuk sama dengan kompleksitas rata-rata, yaitu sebagai berikut:

$$O(n^2 \log n)$$

2. Analisis hasil uji

Berdasarkan hasil pengujian, terdapat beberapa poin *insight* yang didapatkan yaitu sebagai berikut:

- Semakin besar nilai *threshold* maka semakin besar tingkat kompresi dan semakin sedikit variasi warna yang ada di gambar dan juga gambar terlihat semakin *blocky*. Selain itu, karena algoritma lebih permisif terhadap perbedaan warna, maka kedalaman maksimal pohon cenderung lebih sedikit yang tentunya juga berkorespondensi dengan jumlah simpul yang juga cenderung lebih sedikit. Hal yang sebaliknya berlaku untuk semakin kecilnya *threshold*.
- Semakin besar nilai *minimum block size* yang dimasukkan, maka semakin besar ukuran area terkecil yang bisa diproses sehingga detail halus pada gambar semakin hilang (semakin *blocky* dan kasar). Selain itu, karena lebih kasar, tingkat kompresi cenderung meningkat karena informasi yang disimpan lebih tidak detail dibandingkan sebelumnya. Hal ini juga menyebabkan kedalaman maksimal *tree* yang cenderung lebih sedikit karena tertahan oleh ukuran blok yang dapat diproses yang tentunya akan berkorespondensi pada jumlah simpul yang relatif lebih sedikit juga. Hal yang sebaliknya berlaku untuk semakin kecilnya *minimum block size*.
- Penggunaan *error calculation method* yang berbeda juga memberikan perlakuan yang berbeda pada gambar dengan penjelasan sebagai berikut:
 - Untuk metode *variance*, prinsip yang digunakan adalah dengan mengukur seberapa besar variansi setiap piksel terhadap rata-rata warna dalam blok. Jika *threshold* rendah, hanya blok yang sangat homogen yang akan disatukan sehingga gambar akan tersegmentasi dengan halus. Jika *threshold* tinggi, lebih banyak blok yang dianggap homogen sehingga gambar lebih *blurry*.
 - Untuk metode *Mean Absolute Deviation* (MAD), prinsipnya adalah mengukur rata-rata dari penyimpangan absolut tiap piksel terhadap rata-rata nilai piksel. Karena MAD lebih toleran terhadap *outlier* dibandingkan *variance*, maka gambar hasil kompresi cenderung lebih *blending* antar area warna yang agak berbeda dibandingkan metode *variance*. Metode ini bisa menjaga bentuk-bentuk kasar dengan tetap meloloskan lebih banyak blok sebagai homogen.
 - Untuk metode *Max Pixel Difference* (MPD), prinsipnya adalah mengambil selisih maksimum antar nilai piksel dalam satu blok. Metode ini sangat sensitif terhadap *outlier*, hanya dibutuhkan satu piksel yang jauh berbeda maka langsung dianggap tidak homogen. Efeknya terhadap hasil kompresi adalah gambar bisa menjadi sangat pecah-pecah jika

threshold terlalu kecil namun cocok untuk menangkap suatu batas objek dalam gambar.

- Untuk metode *entropy*, prinsipnya adalah mengukur kompleksitas distribusi nilai warna. Semakin tinggi entropi maka semakin tidak teratur warna yang ada dalam blok tersebut. Gambar hasil kompresi dengan entropi cenderung menjaga informasi visual terutama di area yang penuh detail. Jika *threshold* terlalu rendah, gambar bisa terpecah halus dan jika terlalu tinggi bisa terlalu kasar dan kehilangan tekstur.
- Untuk metode *Structural Similarity Index (SSIM)*, prinsipnya adalah membandingkan blok terhadap versi homogen untuk mengukur kesamaan struktur, kontras, dan luminansi. Metode ini memperhatikan struktur lokal pada gambar sehingga hasil kompresi terlihat lebih “*visual-friendly*” dibandingkan metode statistik murni seperti keempat metode pertama. Gambar akhir cenderung mempertahankan bentuk visual gambar sehingga tidak *over-smoothed* seperti pada MAD atau *variance*.
- Setelah melakukan banyak percobaan, sangat sulit/hampir tidak bisa untuk mencapai persentase kompresi dengan nilai kurang dari 50 persen. Hal ini disebabkan oleh sifat dasar algoritma yang secara hierarkis membagi citra hanya ketika tingkat non-homogenitas wilayah melampaui ambang batas (*threshold*) yang ditentukan. Bahkan ketika *threshold* disetel ke nilai yang sangat rendah dan ukuran blok minimum diperbesar untuk membatasi kedalaman pemecahan, quadtree tetap akan menyatukan banyak area seragam ke dalam node tunggal. Jadi sebenarnya ini kembali lagi dengan gambar yang digunakan sebagai input, semakin kontras/berbeda warna yang ada di dalamnya, semakin mudah untuk mencapai tingkat kompresi yang rendah dengan penyesuaian *threshold* dan *minimum block size*.

E. Penjelasan implementasi bonus

1. Penjelasan bonus *Structural Similarity Index (SSIM)*

SSIM merupakan salah satu metode untuk mengukur kemiripan antara dua gambar. Dalam konteks kompresi gambar QuadTree, *SSIM* akan mengukur kemiripan blok gambar asli dengan blok hasil kompresi yang direpresentasikan sebagai blok dengan warna rata-rata. Nilai *SSIM* yang mendekati 1 menunjukkan bahwa blok dapat direpresentasikan oleh warna rata-rata sehingga blok tersebut akan dianggap seragam dan tidak perlu dibagi lagi. Tahapan dari perhitungan *SSIM* adalah sebagai berikut :

- a. Menghitung rata-rata warna dari suatu blok pada gambar
- b. Membentuk suatu uniform blok dengan warna rata-rata tersebut
- c. Menghitung nilai *SSIM* antara blok gambar original dengan blok uniform

d. Mengembalikan true jika nilai kemiripannya diatas threshold

ALGORITMA

```
function calculateSSIM(){menggunakan formula yang tertera pada Gambar3}

function ssim (image, x, y, width, height, threshold, mean)
    {menghitung rata-rata blok}
    i traversal [y...height]
        i traversal [x... width]
            sumR ← sumR + image[j][i].r
            sumG ← sumG + image[j][i].g
            sumB ← sumB + image[j][i].b
            count ← count +1
    mean.r ← sumR / count
    mean.g ← sumG / count
    Mean.b ← sumB / count

    {buat blokk uniform}
    uniformBlok[height, <width, mean>]

    {menghitung nilai SSIM}
    ssimValue ← calculateSSIM(image, uniformBlok, x, y, width,
    height)

    → ssimValue > threshold
```

2. Penjelasan bonus target persentase kompresi

Tujuan utama bonus ini adalah untuk memberikan gambar hasil kompresi yang sedekat mungkin dengan target persentase kompresi yang diinginkan oleh pengguna. Mode ini akan aktif ketika pengguna menuliskan nilai target kompresi selain nol. Untuk bonus ini, diimplementasikan dalam dua fungsi yang mengembalikan nilai *threshold (float)* yang menghasilkan tingkat kompresi yang paling mendekati target. Alasan dibaliknya mengapa ada dua fungsi karena adanya bonus SSIM yang membutuhkan perlakuan khusus karena:

- Prosedur buildTree() untuk kompresi dengan SSIM berbeda dengan buildTree yang digunakan untuk kompresi non-SSIM karena SSIM membutuhkan argumen tambahan.
- Logika dari penilaian error yang digunakan SSIM berbeda dengan non-SSIM. Untuk non-SSIM, semakin kecil nilai *threshold*, semakin kecil pula persentase kompresi yang dihasilkan, hal yang berbalik berlaku untuk SSIM.

Berikut adalah *pseudocode* dari kedua fungsi

```
FUNCTION standardPercentageCompression(image, inputImagePath, outputImagePath,
    errorMethod, minBlockSize, targetCompression, maxDepth, nodeCount):

    low ← lowestThreshold(errorMethod)
```



```

high ← highestThreshold(errorMethod)
bestThreshold ← high
tolerance ← 0.01
precision ← 0.001

WHILE (high - low > precision):
    processedImage ← copy of image
    mid ← (low + high) / 2

    CREATE QuadTree qt
    qt.buildTree(processedImage, mid, errorMethod, minBlockSize)
    qt.reconstructImage(processedImage)

    saveCompressedImage(processedImage, outputPath)

    inputSize ← getFileSize(inputImagePath)
    outputSize ← getFileSize(outputImagePath)
    achievedCompression ← 1 - (outputSize / inputSize)

    diff ← ABS(achievedCompression - targetCompression)

    IF diff <= tolerance:
        bestThreshold ← mid
        maxDepth ← qt.maxDepth
        nodeCount ← qt.nodeCount
        BREAK

    IF achievedCompression > targetCompression:
        high ← mid
    ELSE:
        low ← mid

    bestThreshold ← mid
    maxDepth ← qt.maxDepth
    nodeCount ← qt.nodeCount

RETURN bestThreshold

FUNCTION ssimPercentageCompression(image1, image2, inputImagePath,
outputImagePath,errorMethod, minBlockSize,targetCompression, maxDepth,
nodeCount):

    low ← 0.0
    high ← 1.0
    bestThreshold ← high
    tolerance ← 0.01
    precision ← 0.001

    WHILE (high - low > precision):
        processedImage ← copy of image1
        mid ← (low + high) / 2

        CREATE QuadTree qt
        qt.buildTree(processedImage, image2, mid, errorMethod, minBlockSize)
        qt.reconstructImage(processedImage)

        saveCompressedImage(processedImage, outputPath)

```

```

inputSize ← getFileSize(inputImagePath)
outputSize ← getFileSize(outputImagePath)
achievedCompression ← 1 - (outputSize / inputSize)

diff ← ABS(achievedCompression - targetCompression)

IF diff ≤ tolerance:
    bestThreshold ← mid
    maxDepth ← qt.maxDepth
    nodeCount ← qt.nodeCount
    BREAK

IF achievedCompression < targetCompression:
    high ← mid // Kompresi belum cukup → turunkan threshold
ELSE:
    low ← mid // Kompresi terlalu tinggi → naikan threshold

bestThreshold ← mid
maxDepth ← qt.maxDepth
nodeCount ← qt.nodeCount

RETURN bestThreshold

```

Fungsi `standardPercentageCompression()` digunakan untuk *error method* selain SSIM sedangkan fungsi `ssimPercentageCompression()` digunakan khusus *error method* SSIM. Secara umum, langkah algoritma dari kedua fungsi sebagai berikut:

- Meng-assign nilai *threshold* terendah dan tertinggi yang mungkin dari *errorMethod* yang digunakan.
- Fungsi mulai memasuki *while* loop yang akan berhenti ketika nilai tengah (*threshold*) yang digunakan saat ini sudah mendekati nilai *threshold* terendah atau *threshold* tertinggi yang mungkin dengan nilai presisi sebesar 0.001.
- Mengambil nilai tengah antara nilai terendah dan tertinggi saat ini
- Membangun dan menjalankan rekursi `buildTree()` dengan nilai tengah sebagai *threshold* yang digunakan
- Menyimpan hasil kompresi dan membaca hasil persentase kompresi.
- Jika persentase kompresi belum sesuai target:
 - Persentase hasil > target kompresi:
 - Jika tidak menggunakan SSIM, maka variabel yang menyimpan batas tertinggi di-assign dengan nilai tengah saat ini
 - Jika menggunakan SSIM, maka variabel yang menyimpan batas terendah di-assign dengan nilai tengah saat ini
 - Persentase hasil < target kompresi:
 - Jika tidak menggunakan SSIM, maka variabel yang menyimpan batas terendah di-assign dengan nilai tengah saat ini
 - Jika menggunakan SSIM, maka variabel yang menyimpan batas tertinggi di-assign dengan nilai tengah saat ini

- Jika persentase kompresi saat ini sudah menghampiri target dengan toleransi sebesar 0.01 (1%) maka *loop* akan secara otomatis berhenti dan fungsi akan mengembalikan nilai *threshold* saat ini. Jika tidak, atau *threshold* saat ini belum mendekati batas atas atau bawah *threshold* makan, maka fungsi akan lanjut melakukan *looping* hingga salah satu kondisi pemutus terjadi.

F. Lampiran

Tautan *repository* gitHub : [numshv/Tucil2_13523058_13523080](https://github.com/numshv/Tucil2_13523058_13523080)

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	✓	

Library yang digunakan untuk mengekstrak nilai RGB dari file gambar:

https://github.com/bulletphysics/bullet3/tree/master/examples/ThirdPartyLibs/stb_image