

Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II tahun 2024/2025
Kompresi Gambar Dengan Metode Quadtree
Julius Arthur - 13523030

1. Penjelasan Algoritma Divide and Conquer

Algoritma divide and conquer adalah pendekatan pemecahan masalah yang bekerja dengan membagi sebuah permasalahan besar yang sulit dipecahkan menjadi permasalahan yang lebih kecil dan lebih mudah diselesaikan. Proses ini terbagi menjadi 3 tahap (*divide*, *conquer*, dan *combine*). Pada tahap *divide*, permasalahan dibagi menjadi beberapa permasalahan kecil. Setelah tahap *divide*, setiap permasalahan diselesaikan satu persatu secara terpisah (*conquer*). Hasil dari setiap permasalahan kecil digabungkan menjadi solusi akhir (*combine*). Algoritma ini banyak digunakan dalam masalah sorting (merge sort dan quick sort), binary search, serta perhitungan matematis.

Dengan membagi permasalahan, algoritma ini mampu meningkatkan efisiensi dan mengurangi kompleksitas pemecahan masalah.

2. QuadTree

Quadtree merupakan sebuah struktur data pohon yang digunakan untuk membagi sebuah data dua dimensi menjadi 4 daerah lebih kecil. Quadtree banyak digunakan dalam pengolahan gambar. Pada quadtree, setiap simpul memiliki 4 buah anak, kecuali simpul daun. Dalam pemrosesan gambar, sebuah simpul merepresentasikan bagian dari gambar yang akan diproses. Simpul menyimpan informasi penting yang membuat pemrosesan gambar menjadi lebih cepat, seperti koordinat serta intensitas tiap warna. Sebuah simpul akan terus dibagi menjadi empat secara rekursif hingga mencapai kondisi minimum.

3. Penerapan Quadtree dan Divide and Conquer pada program

Program kompresi gambar ini menggunakan quadtree secara rekursif untuk mengimplementasikan algoritma divide and conquer. Program ini bertujuan untuk menurunkan ukuran gambar secara lossy (menghilangkan detail gambar). Terdapat dua nilai yang menjadi kondisi basis rekursif Quadtree, yaitu keseragaman warna dan jumlah blok minimum. Pendekatan ini mampu menurunkan ukuran data gambar tanpa

menurunkan kualitas gambar secara signifikan terutama pada area dengan warna yang seragam.

Pembagian simpul menjadi empat bagian dihentikan saat area pada simpul memiliki variansi warna (error) dibawah threshold atau jumlah blok pada simpul lebih kecil daripada jumlah minimum blok.

Source Code

Program ini menggunakan bahasa C++. Untuk pemrosesan gambar, digunakan library opencv. Data intensitas warna gambar disimpan dalam array yang berisi uchar (uchar*). Warna disimpan berurutan, yakni biru, hijau, dan merah dari kiri atas hingga kanan bawah gambar.

Error.cpp

```
double Error::average(unsigned char* data, int i_col, int channel, int row0, int col0, int row1, int col1){
    long double res = 0;
    int idx;
    double value;
    double x;
    int n = 0;
    for (int i = row0; i <= row1; i++){
        for (int j = col0; j <= col1; j++){
            n++;
            idx = (i * i_col + j) * 3;
            value = static_cast<double>(data[idx + channel]);
            res += value;
            // cout<<res<<endl;
        }
    }
    // cout<<"x: "<<x;
    // cout<<"res " <<res<<endl;
    return (res/n);
}

double Error::variance(unsigned char* data, int i_col, int channel, int row0, int col0, int row1, int col1){
    double res = 0;
    double val;
    double avg = average(data, i_col, channel, row0, col0, row1, col1);
    int idx = 0;
    int n = 0;
    for (int i = row0; i <= row1; i++){
        for (int j = col0; j <= col1; j++){
            n++;
            idx = (i * i_col + j) * 3;
```

```

        val = static_cast<double>(data[idx + channel]);
        res += pow((val - avg), 2);
    }
}

return res / n;
}

double Error::MAD(unsigned char* data, int i_col, int channel, int row0,
int col0, int row1, int col1){
    double avg = average(data, i_col, channel, row0, col0, row1, col1);
    double res = 0;
    int n = 0;
    int idx;
    for (int i = row0; i <= row1; i++){
        for (int j = col0; j <= col1; j++){
            n++;
            idx = (i * i_col + j) * 3;
            double val = static_cast<double>(data[idx + channel]);
            res += fabs((val - avg));
        }
    }
    return res / n;
}

double Error::MPD(unsigned char* data, int i_col, int channel, int row0,
int col0, int row1, int col1){
    double max = data[0];
    double min = data[0];
    int n = 0;
    int idx;

    for (int i = row0; i <= row1; i++){
        for (int j = col0; j <= col1; j++){
            n++;
            idx = (i * i_col + j) * 3;
            double val = static_cast<double>(data[idx + channel]);
            if (val > max){
                max = val;
            }
            if (val < min){

```

```

        min = val;
    }
}
}

return (max - min);
}

double Error::entropy(unsigned char* data, int i_col, int channel, int row0,
int col0, int row1, int col1){
    std::vector<double> table(256, 0);
    double prob;
    double res = 0;
    int idx;
    int n = 0;

    for (int i = row0; i <= row1; i++){
        for (int j = col0; j <= col1; j++){
            n++;
            idx = (i * i_col + j) * 3;
            int val = static_cast<int>(data[idx + channel]);
            table[val] += 1;
        }
    }

    for (int i = row0; i <= row1; i++){
        for (int j = col0; j <= col1; j++){
            idx = (i * i_col + j) * 3;
            int val = static_cast<int>(data[idx + channel]);
            prob = table[val] / n;
            res += prob * log2(prob);
        }
    }

    return ((-1) * res);
}

```

QuadNode.hpp

#ifndef QUAD_NODE_HPP

```

#define QUAD_NODE_HPP
// #include <opencv2/opencv.hpp>

class QuadNode{
public:
    int row0;
    int row1;
    int col0;
    int col1;
    int depth;
    QuadNode* child[4];

    QuadNode(int row0, int row1, int col0, int col1, int depth);
    QuadNode(QuadNode&);
    ~QuadNode();
    void operator=(QuadNode&);
    void split();

};

#endif

```

QuadNode.cpp

```

#include "QuadNode.hpp"

QuadNode::QuadNode(int row0, int row1, int col0, int col1, int depth){
    this->row0 = row0;
    this->row1 = row1;
    this->col0 = col0;
    this->col1 = col1;
    this->depth = depth;
    for (int i = 0; i < 4; i++){
        child[i] = nullptr;
    }
}

QuadNode::QuadNode(QuadNode& other){
    this->row0 = other.row0;
    this->row1 = other.row1;
}

```

```

this->col0 = other.col0;
this->col1 = other.col1;
this->depth = other.depth;
}

void QuadNode::operator=(QuadNode& other){
    this->row0 = other.row0;
    this->row1 = other.row1;
    this->col0 = other.col0;
    this->col1 = other.col1;
    this->depth = other.depth;
}

void QuadNode::split(){
    int row_mid = row0 + (row1 - row0)/2;
    int col_mid = col0 + (col1-col0) / 2;
    int child_depth = depth + 1;
    child[0] = new QuadNode(row0, row_mid, col0, col_mid, child_depth);
    child[1] = new QuadNode(row0, row_mid, col_mid, col1, child_depth);
    child[2] = new QuadNode(row_mid, row1, col0, col_mid, child_depth);
    child[3] = new QuadNode(row_mid, row1, col_mid, col1, child_depth);

}

QuadNode::~QuadNode(){
    for (int i =0; i < 4; i++){
        delete child[i];
        child[i] = nullptr;
    }
}

```

QuadTree.hpp

```

#ifndef QUAD_TREE_HPP
#define QUAD_TREE_HPP
#include "QuadNode.hpp"
#include "Error.hpp"

```

```
#include "string.h"

class QuadTree{
public:
    QuadNode* root;
    int min_block;
    double error_thres;
    int rows;
    int cols;
    unsigned char* image;
    unsigned char* ori;
    int error_calc;
    int depth = 0;
    int leaf = 0;
    QuadTree(int row0, int row1, int col0, int col1, unsigned char*
image,int rows,int cols, int min_block, double error, int error_calc);
    ~QuadTree();
    void BuildTree(QuadNode* node);
    void CreateGif(int depth, unsigned char* gif, QuadNode* node);
};

#endif
```

QuadTree.cpp

```
#include "QuadTree.hpp"

QuadTree::QuadTree(int row0, int row1, int col0, int col1, unsigned char*
image, int rows, int cols, int min, double error, int e){
    root = new QuadNode(row0, row1, col0, col1, 0);
    this->image = new unsigned char[rows * cols * 3];
    memcpy(this->image, image, rows * cols * 3);
    this->rows = rows;
    this->cols = cols;
    this->min_block = min;
    this->error_thres = error;
    this->error_calc = e;
    this->ori = new unsigned char[rows * cols * 3];
    memcpy(this->ori, image, rows*cols * 3);
}

QuadTree::~QuadTree(){
```

```

        delete[] image;
        delete[] ori;
    }

void QuadTree::BuildTree(QuadNode* node){
    // cout<<"tree\n";
    if (node->depth > depth){
        depth = node->depth;
    }
    double varB, varG, varR;
    if (error_calc == 1){
        varB= Error::variance(image, cols, 0, node->row0, node->col0,
node->row1, node->col1);
        varG = Error::variance(image, cols, 1, node->row0, node->col0,
node->row1, node->col1);
        varR = Error::variance(image, cols, 2, node->row0, node->col0,
node->row1, node->col1);
    }

    else if (error_calc == 2){
        varB = Error::MAD(image, cols, 0, node->row0, node->col0,
node->row1, node->col1);
        varG = Error::MAD(image, cols, 1, node->row0, node->col0,
node->row1, node->col1);
        varR = Error::MAD(image, cols, 2, node->row0, node->col0,
node->row1, node->col1);
    }

    else if (error_calc == 3){
        varB = Error::MPD(image, cols, 0, node->row0, node->col0,
node->row1, node->col1);
        varG = Error::MPD(image, cols, 1, node->row0, node->col0,
node->row1, node->col1);
        varR = Error::MPD(image, cols, 2, node->row0, node->col0,
node->row1, node->col1);
    }

    else if (error_calc == 4){
        varB = Error::entropy(image, cols, 0, node->row0, node->col0,
node->row1, node->col1);
        varG = Error::entropy(image, cols, 1, node->row0, node->col0,
node->row1, node->col1);
    }
}

```

```

    varR = Error::entropy(image, cols, 2, node->row0, node->col0,
node->row1, node->col1);
}

double var = (varB + varG + varR)/3;
int size = (node->row1 - node->row0) * (node->col1 - node->col0);
bool valid = false;

if (var > error_thres){
    if (size > min_block && size/4 >= min_block){
        valid = true;
        node->split();
        leaf += 4;
        for (int i = 0; i < 4; i++){
            BuildTree(node->child[i]);
        }
    }
}

if (!valid){
    double avgB = Error::average(image, cols, 0, node->row0,
node->col0, node->row1, node->col1);
    double avgG = Error::average(image, cols, 1, node->row0,
node->col0, node->row1, node->col1);
    double avgR = Error::average(image, cols, 2, node->row0,
node->col0, node->row1, node->col1);
    int idx;

    for (int i = node->row0; i < node->row1; i++){
        for (int j = node->col0; j < node->col1; j++){
            idx = (i * cols + j) * 3;
            image[idx+0] = static_cast<uint8_t>(avgB);
            image[idx+1] = static_cast<uint8_t>(avgG);
            image[idx+2] = static_cast<uint8_t>(avgR);
        }
    }
}
}

void QuadTree::CreateGif(int depth_target, unsigned char* gif,

```

```

QuadNode* node){
    bool isLeaf = true;
    if (node->depth < depth_target){
        if (node->child[0] != nullptr){
            isLeaf = false;
            CreateGif(depth_target, gif, node->child[0]);
            CreateGif(depth_target, gif, node->child[1]);
            CreateGif(depth_target, gif, node->child[2]);
            CreateGif(depth_target, gif, node->child[3]);
        }
    }

    if (isLeaf){
        double avgR = Error::average(ori, cols, 0, node->row0, node->col0,
node->row1, node->col1);
        double avgG = Error::average(ori, cols, 1, node->row0, node->col0,
node->row1, node->col1);
        double avgB = Error::average(ori, cols, 2, node->row0, node->col0,
node->row1, node->col1);
        int idx;

        for (int i = node->row0; i < node->row1; i++){
            for (int j = node->col0; j < node->col1; j++){
                idx = (i * cols + j) * 3;
                gif[idx+0] = static_cast<uint8_t>(avgR); // R
                gif[idx+1] = static_cast<uint8_t>(avgG); // G
                gif[idx+2] = static_cast<uint8_t>(avgB); // B
            }
        }
    }
}

```

Langkah - langkah algoritma pada program:

1. Program menginisiasi sebuah pohon Quadtree. Akar/root pohon merepresentasikan input gambar secara keseluruhan (koordinat (0,0) dan panjang serta lebar yang disimpan pada simpul sama dengan gambar secara keseluruhan.

Konstruktor Quadtree

```
Function QuadTreeConstructor(row0, row1, col0, col1, image,
rows, cols, min, error, e):
    root ← new QuadNode(row0, row1, col0, col1, depth = 0)
    this.image ← image
    this.rows ← rows
    this.cols ← cols
    this.min_block ← min
    this.error_thres ← error
    this.error_calc ← e
    this.ori ← image
EndFunction
```

Tree diinisialisasi dengan row0 dan col0 bernilai 0 serta row1 dan col1 adalah tinggi dan lebar gambar secara berurutan. Data image pada parameter di *copy* dengan *deep copy*.

2. Program menghitung variansi warna (error) pada area gambar tersebut. Error dihitung dengan 4 cara yaitu varians, *Mean Absolute Deviation*, *Max Pixel Difference*, dan *Entropy*. Setiap metode akan menghasilkan sebuah nilai error/ keberagaman warna sebuah area.

Fungsi BuildTree

```
FUNCTION BuildTree(node)
    // Perbarui nilai kedalaman maksimum jika diperlukan
    IF node.depth > depth THEN
        depth ← node.depth
    END IF

    DECLARE varB, varG, varR
```

```

// Hitung error berdasarkan mode perhitungan error (variance,
MAD, MPD, atau entropy)
IF error_calc == 1 THEN
    varR ← Error.variance(image, cols, channel=0, node.row0,
node.col0, node.row1, node.col1)
    varG ← Error.variance(image, cols, channel=1, node.row0,
node.col0, node.row1, node.col1)
    varB ← Error.variance(image, cols, channel=2, node.row0,
node.col0, node.row1, node.col1)

ELSE IF error_calc == 2 THEN
    varR ← Error.MAD(image, cols, channel=0, node.row0,
node.col0, node.row1, node.col1)
    varG ← Error.MAD(image, cols, channel=1, node.row0,
node.col0, node.row1, node.col1)
    varB ← Error.MAD(image, cols, channel=2, node.row0,
node.col0, node.row1, node.col1)

ELSE IF error_calc == 3 THEN
    varR ← Error.MPD(image, cols, channel=0, node.row0,
node.col0, node.row1, node.col1)
    varG ← Error.MPD(image, cols, channel=1, node.row0,
node.col0, node.row1, node.col1)
    varB ← Error.MPD(image, cols, channel=2, node.row0,
node.col0, node.row1, node.col1)

ELSE IF error_calc == 4 THEN
    varR ← Error.entropy(image, cols, channel=0, node.row0,
node.col0, node.row1, node.col1)
    varG ← Error.entropy(image, cols, channel=1, node.row0,
node.col0, node.row1, node.col1)
    varB ← Error.entropy(image, cols, channel=2, node.row0,
node.col0, node.row1, node.col1)
END IF

// Rata-rata error dari ketiga channel
var_total ← (varB + varG + varR) / 3

```

3. Kemudian, rata - rata error dari 3 warna (RGB) akan dibandingkan dengan nilai threshold. Jika rata - rata error hasil perhitungan lebih kecil daripada threshold dan jumlah blok saat ini dan setelah pembagian masih diatas jumlah minimum blok, simpul akan dibagi (*split*), sehingga memiliki 4 buah anak. Jumlah kedalaman akan bertambah dan simpul bertambah 4 buah.

```

// Hitung ukuran area node
size ← (node.row1 - node.row0) * (node.col1 - node.col0)
valid ← FALSE

// Periksa apakah area memenuhi kriteria error threshold dan
ukuran minimal
IF var_total > error_thres THEN
    IF size > min_block AND (size / 4) >= min_block THEN
        valid ← TRUE
        // Bagi node menjadi empat anak
        node.split()
        leaf ← leaf + 4 // update jumlah node daun
        FOR i FROM 0 TO 3 DO
            BuildTree(node.child[i])
        END FOR
    END IF
END IF

```

4. Proses *split* dilakukan hingga tidak memenuhi kondisi melakukan *split*. Pada kondisi ini, rekursi dihentikan dan dilakukan tahap normalisasi. Warna di blok pada posisi yang sesuai dengan simpul akan diisi dengan rata - rata warna pada simpul tersebut. Akibatnya, seluruh blok pada simpul akan memiliki nilai intensitas warna yang sama.

```

// Jika node tidak memenuhi kriteria pembagian, isi blok dengan
nilai rata-rata
IF valid == FALSE THEN
    avgR ← Error.average(image, cols, channel=0, node.row0,
node.col0, node.row1, node.col1)
    avgG ← Error.average(image, cols, channel=1, node.row0,
node.col0, node.row1, node.col1)

```

```
avgB ← Error.average(image, cols, channel=2, node.row0,  
node.col0, node.row1, node.col1)  
  
FOR i FROM node.row0 TO node.row1 DO  
    FOR j FROM node.col0 TO node.col1 DO  
        idx ← (i * cols + j) * 3  
        image[idx + 0] ← avgR  
        image[idx + 1] ← avgG  
        image[idx + 2] ← avgB  
    END FOR  
END FOR  
END IF  
END FUNCTION
```

Tes 1:

```
File (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/before/tc1.jpg

Pilih perhitungan error (masukkan nomor):
1. Variance
2. Mean absolute deviation
3. Max Pixel Difference
4. Entropy
1

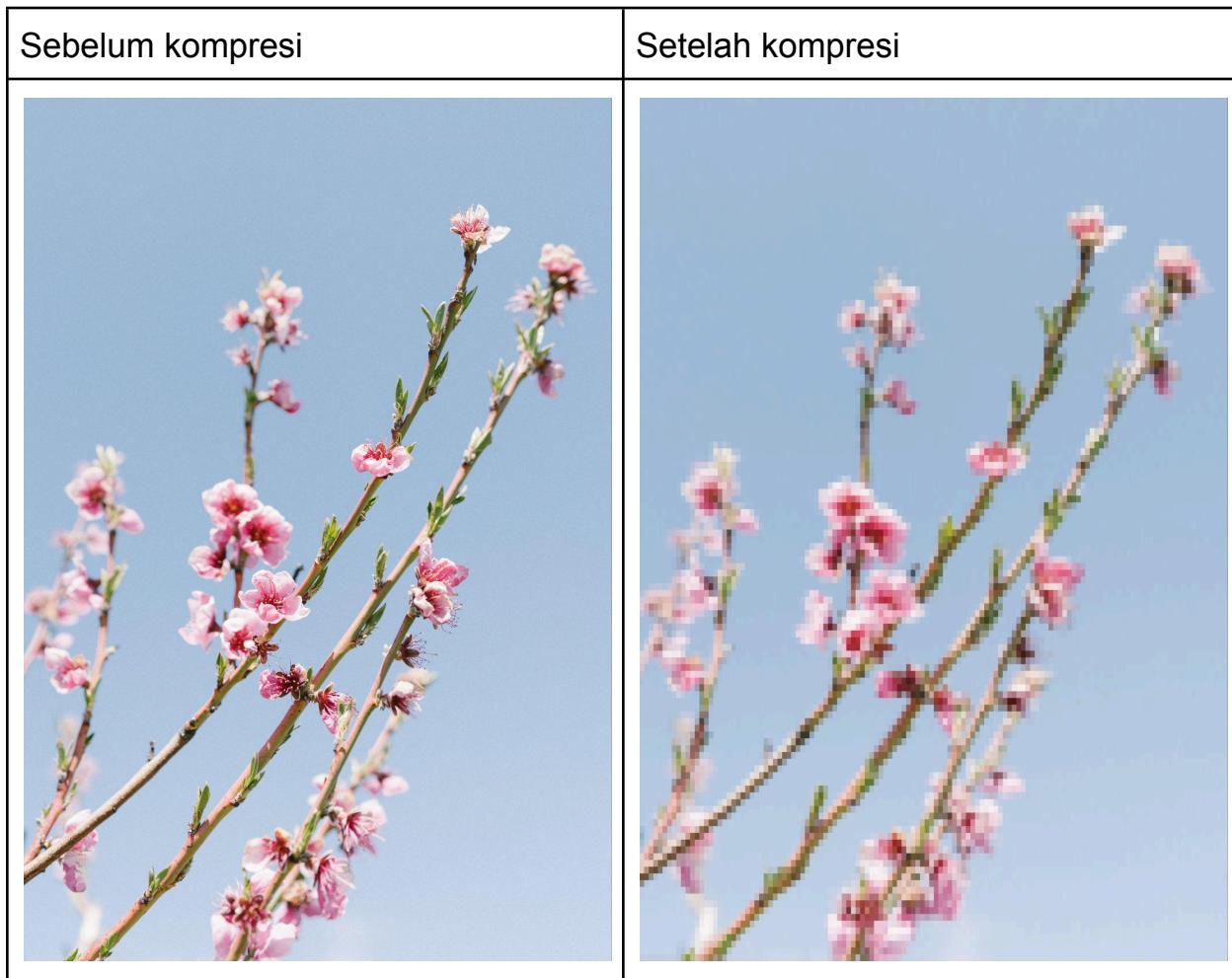
Error: 20

Min_block: 500

output gambar (absolut dan dengan ekstensi): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/tc1.jpg

output gif (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/gif1.gif

Waktu eksekusi: 116361 ms
Ukuran awal: 2.66752e+06 bytes.
Ukuran akhir: 1.98641e+06 bytes.
Persentase kompresi: 25.5334%
Kedalaman pohon: 7
Banyak simpul: 19668
```



Tes 2:

```
julius@ArthurPC:/mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/src$ ./main2
File (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/before/tc2.jpg

Pilih perhitungan error (masukkan nomor):
1. Variance
2. Mean absolute deviation
3. Max Pixel Difference
4. Entropy
2

Error: 8

Min_block: 100

Output gambar (absolut dan dengan ekstensi): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/tc2.jpg

Output gif (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/gif2.gif

Waktu eksekusi: 93139 ms
Ukuran awal: 2.47169e+06 bytes.
Ukuran akhir: 4.7054e+06 bytes.
Persentase kompresi: -90.372%
Kedalaman pohon: 8
Banyak simpul: 77324
```

Sebelum kompresi	Setelah kompresi
	

Tes 3

```
File (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/before/tc3.jpg  
Pilih perhitungan error (masukkan nomor):  
1. Variance  
2. Mean absolute deviation  
3. Max Pixel Difference  
4. Entropy  
3  
Error: 5  
Min_block: 1000  
Output gambar (absolut dan dengan ekstensi): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/tc3.jpg  
Output gif (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/gif3.gif  
Waktu eksekusi: 114031 ms  
Ukuran awal: 3.31475e+06 bytes.  
Ukuran akhir: 3.20576e+06 bytes.  
Persentase kompresi: 3.28815%  
Kedalaman pohon: 7  
Banyak simpul: 21844
```



Tes 4

```
julius@ArthurPC:/mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/src$ ./main2  
File (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/before/tc4.jpg  
  
Pilih perhitungan error (masukkan nomor):  
1. Variance  
2. Mean absolute deviation  
3. Max Pixel Difference  
4. Entropy  
4  
  
Error: 10  
  
Min_block: 2000  
  
Output gambar (absolut dan dengan ekstensi): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/tc4.jpg  
Output gif (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/gif4.gif  
  
Waktu eksekusi: 84902 ms  
Ukuran awal: 1.32223e+06 bytes.  
Ukuran akhir: 1.63722e+06 bytes.  
Persentase kompresi: -23.8227%  
Kedalaman pohon: 6  
Banyak simpul: 5460
```

Sebelum kompresi	Setelah kompresi
	

Tes 5

```
julius@ArthurPC:/mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/src$ ./main2
File (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/before/tc5.jpg

Pilih perhitungan error (masukkan nomor):
1. Variance
2. Mean absolute deviation
3. Max Pixel Difference
4. Entropy
7
Pilih method 1 - 4
Method: 1

Error: 20

Min_block: 3000

Output gambar (absolut dan dengan ekstensi): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/tc5.pdf

Ekstensi tidak valid
File:/mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/tc5.jpg

Output gif (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/gif5.gif

Waktu eksekusi: 60423 ms
Ukuran awal: 1.83486e+06 bytes.
Ukuran akhir: 1.75032e+06 bytes.
Persentase kompresi: 4.60755%
Kedalaman pohon: 6
Banyak simpul: 5372
```

Sebelum kompresi	Setelah kompresi
	

Tes 6

```
julius@ArthurPC:/mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/src$ ./main2
File (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/before/tc6.jpg

Pilih perhitungan error (masukkan nomor):
1. Variance
2. Mean absolute deviation
3. Max Pixel Difference
4. Entropy
2

Error: 20

Min_block: 3000

Output gambar (absolut dan dengan ekstensi): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/tc6.jpg

Output gif (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/gif6.gif

Waktu eksekusi: 60848 ms
Ukuran awal: 1.96474e+06 bytes.
Ukuran akhir: 1.09025e+06 bytes.
Persentase kompresi: 44.508%
Kedalaman pohon: 6
Banyak simpul: 1920
```



Tes 7

```
julius@ArthurPC:/mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/src$ ./main2
File (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/before/tc7.jpg

Pilih perhitungan error (masukkan nomor):
1. Variance
2. Mean absolute deviation
3. Max Pixel Difference
4. Entropy
3

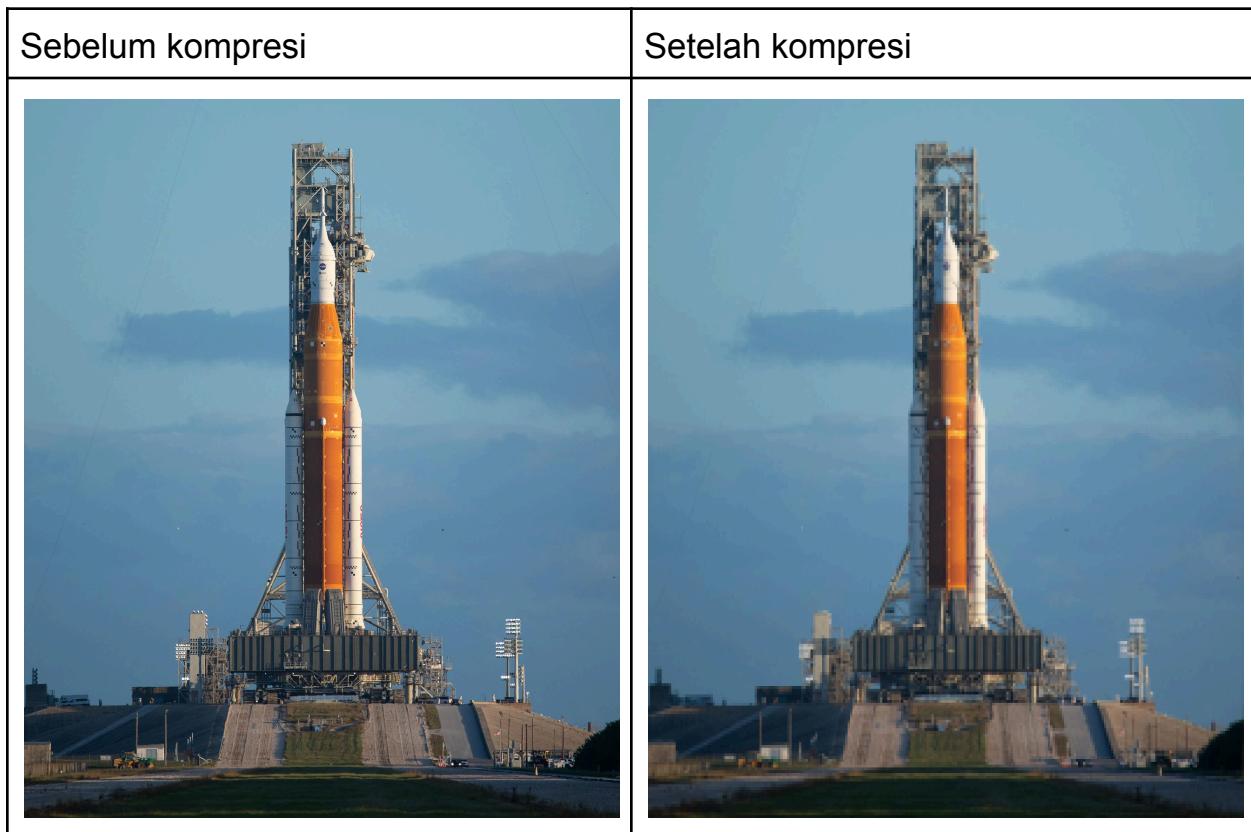
Error: 5

Min_block: 50

Output gambar (absolut dan dengan ekstensi): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/tc7.jpg

Output gif (absolut): /mnt/c/Users/User/Documents/ITB/Stima/Tucil2/Tucil2_13523030/test/gif7.gif

Waktu eksekusi: 16628 ms
Ukuran awal: 618541 bytes.
Ukuran akhir: 598942 bytes.
Persentase kompresi: 3.1685%
Kedalaman pohon: 8
Banyak simpul: 87380
```



Analisis Algoritma Divide and Conquer Menggunakan Quadtree

Algoritma pada Quadtree memanfaatkan konsep divide and conquer untuk membagi gambar menjadi blok lebih kecil berdasarkan keragaman warna. Setiap blok akan dibagi menjadi empat bagian berdasarkan nilai keseragaman warna (varians, MAD, MPD, atau entropi), dan jumlah blok itu sendiri. Proses dilakukan secara rekursif hingga semua blok tidak dapat dibagi lebih kecil lagi.

Divide and conquer efektif diterapkan pada kasus kompresi gambar karena mampu memproses bagian gambar yang kompleks dan menghasilkan gambar yang lebih seragam. Blok yang sudah seragam akan berhenti diproses dan blok yang kompleks akan terus diproses. Maka, dalam hal ini, struktur pohon, seperti Quadtree juga dapat bekerja dengan efisien untuk memproses gambar/citra secara rekursif.

Perhitungan Error dilakukan pada setiap simpul, pada setiap blok. Jika ukuran blok adalah $k \times k$, maka, perhitungan membutuhkan waktu $O(k^2)$. Sedangkan, normalisasi dilakukan setiap sebuah simpul menjadi daun. Setiap pixel akan dihitung, sehingga akan memiliki kompleksitas $O(k^2)$.

Kompleksitas algoritma bergantung pada keberagaman warna pixel pada gambar. Pada kasus terbaik, setiap pixel memiliki keragaman dibawah threshold, sehingga hanya terjadi sekali perhitungan error. Maka, kompleksitasnya adalah $O(n^2)$. Kasus terburuk terjadi saat gambar dibagi empat terus menerus hingga didapat pixel berukuran (1×1) , sehingga terdapat n^2 simpul. Pada suatu level k , terdapat blok berukuran $(n/2^k \times n/2^k)$. Maka, seluruh pekerjaan pada level k memiliki kompleksitas $4^k \times O((n/2^k)^2) = O(n^2)$. Lalu, kedalaman pohon bergantung pada threshold dan jumlah minimum blok, secara umum sebanyak $(\log n)$.

Maka secara umum, algoritma pada proses kompresi memiliki kompleksitas $\log n \times O(n^2) = O(n^2 \log n)$.

Penerapan Bonus GIF

GIF dari proses ini merupakan kumpulan gambar setiap terjadinya split. Akibatnya, kumpulan gambar yang berurutan akan menunjukkan perkembangan gambar hingga akhir proses kompresi. Proses GIF dimulai dengan meninjau root dari QuadTree. Program menyimpan gambar pada level ini dan melanjutkan proses. Jika terdapat anak, pointer akan pindah ke seluruh anak secara rekursif dan kembali menyimpan gambar pada level ini. Lama waktu proses kompresi terpengaruh oleh proses GIF ini. Semakin banyak node pada pohon, semakin lama pula proses menghasilkan GIF.

Github: https://github.com/julius123123/Tucil2_13523030

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	V	
2. Program berhasil dijalankan	V	
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	V	
4. Mengimplementasi seluruh metode perhitungan error wajib	V	
5. [Bonus] Implementasi persentase kompresi sebagai parameter tambahan		V
6. [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error		V
7. [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	V	
8. Program dan laporan dibuat (kelompok) sendiri	V	