2IF2211 Strategi Algoritma

Laporan Tugas Kecil 2

oleh

Ahmad Wafi Izdzharulhaqq 13523131

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Semester II Tahun 2024/2025

DAFTAR ISI

1. Deskripsi Tugas

1.1. Deskripsi Umum

Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis sistem warna RGB, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika bagian tersebut tidak seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan. Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

1.2. Rincian Spesifikasi

Ide pada tugas kecil 2 ini cukup sederhana, seperti pada pembahasan sebelumnya mengenai Quadtree. Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (Divide and Conquer):

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

a)Metode perhitungan variansi: variance, MAD, MPD, Entropy SSIM

b)Threshold variansi: nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.

c)Minimum block size: ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.

2. Perhitungan Error UnIde pada tugas kecil 2 ini cukup sederhana, seperti pada pembahasan sebelumnya

mengenai Quadtree. Berikut adalah prosedur pada program kompresi gambar yang akan dibuat

dalam Tugas Kecil 2 (Divide and Conquer):

1. Inisialisasi dan Persiapan Data

Masukkan gambar yang akan dikompresi akan diolah dalam format matriks piksel

dengan nilai intensitas berdasarkan sistem warna RGB. Berikut adalah

parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan

kompresi gambar:

IF2211 Strategi Algoritma – Tugas Kecil 2

2

a) Metode perhitungan variansi: pilih metode perhitungan variansi berdasarkan

opsi yang tersedia pada Tabel 1.

b) Threshold variansi: nilai ambang batas untuk menentukan apakah blok akan

dibagi lagi.

c) Minimum block size: ukuran minimum blok piksel yang diperbolehkan untuk

diproses lebih lanjut.

2. Perhitungan Error

Untuk setiap blok gambar yang sedang diproses, hitung nilai variansi menggunakan

metode yang dipilih sesuai Tabel 1.

3. Pembagian Blok

Bandingkan nilai variansi blok dengan threshold:

❖ Jika variansi di atas threshold (cek kasus khusus untuk metode bonus), ukuran

blok lebih besar dari minimum block size, dan ukuran blok setelah dibagi

menjadi empat tidak kurang dari minimum block size, blok tersebut dibagi

menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.

❖ Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan

untuk blok tersebut.

4. Normalisasi Warna

Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai dengan

rata-rata nilai RGB blok.

5. Rekursi dan Penghentian

Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua

blok memenuhi salah satu dari dua kondisi berikut:

❖ Error blok berada di bawah threshold.

❖ Ukuran blok setelah dibagi menjadi empat kurang dari minimum block size.

6. Penyimpanan dan Output

Rekonstruksi gambar dilakukan berdasarkan struktur QuadTree yang telah dihasilkan

selama proses kompresi. Gambar hasil rekonstruksi akan disimpan sebagai file

terkompresi. Selain itu, persentase kompresi akan dihitung dan disertakan dengan rumus

sesuai dengan yang terlampir pada dokumen ini. Persentase kompresi ini memberikan

gambaran mengenai efisiensi metode kompresi yang digunakan.tuk setiap blok gambar yang sedang diproses, hitung nilai variansi menggunakan metode yang dipilih. Bandingkan nilai variansi blok dengan threshold:

- ❖ Jika variansi di atas threshold (cek kasus khusus untuk metode bonus), ukuran blok lebih besar dari minimum block size, dan ukuran blok setelah dibagi menjadi empat tidak kurang dari minimum block size, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.
- ❖ Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.

4. Normalisasi Warna Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai dengan rata-rata nilai RGB blok. 5. Rekursi dan Penghentian Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua blok memenuhi salah satu dari dua kondisi berikut:

- ❖ Error blok berada di bawah threshold.
- ❖ Ukuran blok setelah dibagi menjadi empat kurang dari minimum block size.

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

2. Landasan Teori

2.1. Algoritma *Divide and Conquer*

2.1.1. Terminologi Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* ditemukan oleh Anatolii Alexeevich Karatsuba pada tahun 1960. Dahulu algoritma ini adalah strategi militer yang dikenal dengan nama divide ut imperes hingga akhirnya strategi ini digunakan dalam ilmu komputer. Algoritma *Divide and Conquer* membuat permasalahan yang besar dipecah menjadi bagian yang lebih kecil untuk memudahkan penyelesaiannya. Dalam konsepnya, sebuah masalah akan diselesaikan per-sub masalahnya. Solusi dari sub masalah tersebut digabungkan dan menjadi solusi akhir dari keseluruhan masalah.

Prinsip dasar dari algoritma ini adalah dengan membagi n input menjadi k subset input yang berbeda ($1<k<n$). Dari k subset input yang berbeda akan terdapat k subproblem. Setiap problem memiliki solusi masing-masing sehingga akan diperoleh k subsolusi. Kemudian dari k subsolusi akan didapat solusi yang diharapkan.

Terdapat 3 langkah-langkah umum dalam algoritma *Divide and Conquer* yaitu :

- Devide : Membagi masalah menjadi beberapa upa-masalah yang mirip masalah semula tapi berukuran lebih kecil.
- Conquer : Menyelesaikan masing-masing upa-masalah secara langsung jika berukuran kecil atau secara rekursif jika berukuran besar.
- Combine : Menggabungkan solusi masing-masing upa-masalah sehingga dapat terbentuk solusi masalah semula.

Dalam penerapannya algoritma Divide and Conquer beberapa persoalan dapat diselesaikan seperti memecahkan Convex Hull, persoalan minimum dan maksimum, optimasi konversi bilangan desimal ke bilangan bulat, dan mengurutkan sekolompok bilangan (merge sort) menjadi kesuluruhan list.

2.2. Quadtree Image Compressor

Quadtree adalah struktur data pohon yang memiliki node internal tepat empat anak. gambar dapat dipecah menjadi blok blok yang lebih kecil dan lebih sederhana. Setiap blok tersebut kemudian dapat diwakili oleh satu simpul dalam struktur quadtree. Pendekatan ini berisiko menghilangkan redundansi dan penyimpanan gambar dengan menggunakan lebih sedikit memori. Metode divide and conquer dengan quadtree dapat juga mengurangi waktu proses transmisi data karena ukuran file yang sudah terkompresi. Quadtrees untuk kompresi gambar membagi gambar secara rekursif menjadi empat subruang, masing-masing memegang warna RGB rata-rata dan kesalahan yang menentukan warna tersebut untuk subruangnya. Ambang batas ditetapkan berdasarkan kesalahan itu dan membantu pohon menentukan apakah simpul harus dibagi lebih lanjut atau tidak.

2.2.1. Image Compression

Image compression adalah proses meminimalisasi jumlah byte yang ada dalam file gambar berfungsi untuk membuat ukuran file gambar menjadi lebih kecil tanpa membuat kualitas gambar di bawah batas awal yang diterima. Umumnya teknik ini digunakan pada proses transmisi data (*data transmission*) dan penyimpanan data (*data storage*). Dalam konteks pengiriman data, file gambar yang telah dikompresi akan mempercepat proses pengiriman data. Lalu dalam konteks penyimpanan, menurunkan ukuran file dapat membuat gambar lebih banyak tersimpan dalam penyimpanan data.

Terdapat dua jenis kompresi gambar, yaitu:

• Lossy compression: yaitu metode yang menghilangkan beberapa informasi dalam citra untuk mengurangi ukuran gambar. Teknik ini bersifat irreversible atau tidak dapat dikembalikan ke gambar semula. Setelah dikompresi file akan kehilangan sebagian kualitasnya. Teknik ini mengubah warna dan detail pada citra nenjadi lebih sederhana tanpa perbedaan mencolok di mata. Penggunaan metode pada kompresi ini misalnya : Transform coding, chroma subsampling, color quantization.Contoh format file dalam kompresi lossy adalah: JPEG, MPEG

• Lossless compression: yaitu metode yang tidak menghilangkan informasi penting dari gambar maupun merusak kualitasnya. Kompresi ini memanfaatkan pola-pola atau redundansi yang ada pada gambar dan memperkecil pola informasi di dalam citra. Penggunaan metode pada kompresi ini adalah entropy encoding dan dictionary based. Contoh format file kompresi lossless adalah : GIF, PCX, BMP, TIFF, TRG, PGM

3. Aplikasi Metode Divide and Conquer

Dalam program ini, sebuah gambar disimpan dalam bentuk vektor dua dimensi dengan elemen berupa data bertipe RGB yang menyimpan nilai red, green, dan blue dari pixel-pixel gambar. Proses kompresi dilakukan dengan memanfaatkan struktur QuadTree yang tersusun atas Node-Node berupa blok partisi gambar dengan menggunakan pendekatan algoritma *divide and conquer*. Setelah mengalami konversi ke dalam vektor,

```
procedure compressImage(input root : pointer to Node , methodChoice : string,
imageLoader : reference of ImageLoader)

{ Menlakukan kompresi gambar dengan algoritma divide and conquer
Masukan: masukan root node, methodCoice, dan alias objek imageLoader
Luaran: solusi data vektor gambar yang sudah terkompresi }

KAMUS LOKAL
    imageData : reference of vector<vector<RGB>>
    rootWidth, rootHeight : integer
    x, y : integer
    error : real
    halfWidth, halfHeight : integer

ALGORITMA
    imageData ← imageLoader.getImageData()
    rootWidth ← root.getNodeWidth()
    rootHeight ← root.getNodeHeight()
    x ← root.getNodeX()
    y ← root.getNodeY()

    root.calculateErrorRate(methodChoice, imageData)
    error ← root.getErrorRate()

    if error > treshold AND rootWidth > minBlockSize AND rootHeight >
minBlockSize AND
       (rootWidth / 2) * (rootHeight / 2) ≥ minBlockSize then

        halfWidth ← rootWidth / 2
        halfHeight ← rootHeight / 2
        numNodes ← numNodes + 4

        { DIVIDE }
        root.setChild(0, new Node(false, x, y, halfWidth, halfHeight,
imageData))              { Top-left }
        root.setChild(1, new Node(false, x + halfWidth, y, halfWidth,
halfHeight, imageData))   { Top-right }
        root.setChild(2, new Node(false, x, y + halfHeight, halfWidth,
halfHeight, imageData))  { Bottom-left }
        root.setChild(3, new Node(false, x + halfWidth, y + halfHeight,
halfWidth, halfHeight, imageData)) { Bottom-right }

        { Rekursif ke empat anak }
        compressImage(root.getChild(0), methodChoice, imageLoader)
        compressImage(root.getChild(1), methodChoice, imageLoader)
        compressImage(root.getChild(2), methodChoice, imageLoader)
        compressImage(root.getChild(3), methodChoice, imageLoader)
```

```
    else
        {CONQUER}
        root.setAsLeaf(true)

        for i ← y to y + rootHeight - 1 do
            for j ← x to x + rootWidth - 1 do
                imageData[i][j].r ← root.getAvgColor().r
                imageData[i][j].g ← root.getAvgColor().g
                imageData[i][j].b ← root.getAvgColor().b
            endfor
        endfor
    endif
endprocedure

{COMBINE: di akhir fungsi, didapati satu vector of vector of RGB yang
merepresentasikan satu gambar utuh sebagai hasil proses divide and conquer
(solusi umum)}
```

Tabel 1. Analisis Elemen *Divide and Conquer*

No	Elemen	Analisis
1	Divide	Dilakukan pada saat awal tahapan kompresi gambar. Proses divide membagi vektor 2 dimensi menjadi partisi yang lebih kecil. Dalam perspektif QuadTree, algoritma divide akan membagi Node menjadi 4 bagian child Nodes yang memiliki ½ panjang dan ½ lebar ukuran Node parent. Setelah itu, akan dilakukan proses divide and conquer untuk setiap child Nodes.
2	Conquer	Apabila memenuhi satu di antara dua kriteria: 1. Hasil perhitungan error rate sudah memenuhi threshold, atau 2. Ukuran node sudah mencapai minimum block size. Maka akan dilakukan operasi dalam membuat solusi lokal, berupa pengubahan nilai RGB atas seluruh elemen pixel dalam suatu node menjadi nilai rata-rata RGB di node tersebut.
3	Combine	Di akhir iterasi akan didapatkan gabungan atas solusi lokal, berupa solusi umum dalam bentuk vector of vector of RGB yang merepresentasikan satu gambar seara keseluruhan.

4. Implementasi dan Pengujian

4.1. Kode Sumber

Repositori Github yang digunakan adalah Tucil2_13523131 . Adapun kode kelas, header dan main dari program ini adalah sebagai berikut (C++):

4.1.1. ImageQuadtree.h

```
#ifndef IMAGE_QUADTREE_H
#define IMAGE_QUADTREE_H
```

```
#include "Node.h"
#include "../utils/ImageLoader.h"
#include <vector>

using namespace std;

class ImageQuadtree {
private:
    const double treshold;
    const double minBlockSize;
    int numNodes;
    int maxDepth;
    Node* root;

public:
    ImageQuadtree(const double treshold, const double minBlockSize,
ImageLoader& loader);
    ~ImageQuadtree();

    void compressImage(Node* root, string methodChoice, ImageLoader& loader);
    void countMaxDepth();
    int countMaxDepthHelper(Node* node, int depth);
    void countNumNodes(Node* node);
    int countNumNodesHelper(Node* node);
    Node* getRoot() const;
    int getTreeDepth() const;
    int getNumNodes() const;


};

#endif
```

4.1.2. ImageQuadtree.cpp

```
#include "ImageQuadtree.h"

ImageQuadtree::ImageQuadtree(const double treshold, const double minBlockSize,
ImageLoader& imageLoader)
    : treshold(treshold),
      minBlockSize(minBlockSize),
      numNodes(1),
      maxDepth(0)
{
    root = new Node(false, 0, 0,
                    imageLoader.getWidth(),
                    imageLoader.getHeight(),
                    imageLoader.getImageData());
}

ImageQuadtree::~ImageQuadtree() {
    if (root != nullptr) {
        root->deleteChildren();
        delete root;
        root = nullptr;
    }
}

void ImageQuadtree::compressImage(Node* root, string methodChoice, ImageLoader&
imageLoader) {
    // 1. hitung error rate
    // 2. hitung ukuran
    // 3. divide? masuk ke rekursif : set semua blok di dalemnya ke avg color
    // 4. combine

    vector<vector<RGB>>& imageData = imageLoader.getImageData();
    int rootWidth = root->getNodeWidth();
```

```
    int rootHeight = root->getNodeHeight();
    int x = root->getNodeX();
    int y = root->getNodeY();

    root->calculateErrorRate(methodChoice, imageData);
    double error = root->getErrorRate();

    if (error > treshold && rootWidth > minBlockSize && rootHeight >
minBlockSize && (rootWidth/2)*(rootHeight/2) >= minBlockSize) {
        int halfWidth = rootWidth / 2;
        int halfHeight = rootHeight / 2;
        numNodes += 4;

        // Buat empat anak root
        root->setChild(0, new Node(false, x, y, halfWidth, halfHeight,
imageData)); // Top-left
        root->setChild(1, new Node(false, x + halfWidth, y, halfWidth,
halfHeight, imageData)); // Top-right
        root->setChild(2, new Node(false, x, y + halfHeight, halfWidth,
halfHeight, imageData)); // Bottom-left
        root->setChild(3, new Node(false, x + halfWidth, y + halfHeight,
halfWidth, halfHeight, imageData)); // Bottom-right

        // Rekursi untuk setiap anak
        compressImage(root->getChild(0), methodChoice, imageLoader);
        compressImage(root->getChild(1), methodChoice, imageLoader);
        compressImage(root->getChild(2), methodChoice, imageLoader);
        compressImage(root->getChild(3), methodChoice, imageLoader);
    }
    else {
        root->setAsLeaf(true);
        for (int i = y; i < y + rootHeight; ++i) {
            for (int j = x; j < x + rootWidth; ++j) {
                imageData[i][j].r = root -> getAvgColor().r;
                imageData[i][j].g = root -> getAvgColor().g;
                imageData[i][j].b = root -> getAvgColor().b;
            }
        }
    }
}

void ImageQuadtree::countMaxDepth() {
    maxDepth = countMaxDepthHelper(root, 0);
}

// Helper function rekursif
int ImageQuadtree::countMaxDepthHelper(Node* root, int currentDepth){
    if (root == nullptr) {
        return currentDepth;
    }

    if (root->isLeafNode()) {
        return currentDepth;
    }

    int maxChildDepth = currentDepth;
    for (int i = 0; i < 4; i++) {
        int childDepth = countMaxDepthHelper(root->getChild(i), currentDepth +
1);
        maxChildDepth = max(maxChildDepth, childDepth);
    }

    return maxChildDepth;
}

void ImageQuadtree::countNumNodes(Node* node){
    numNodes = countNumNodesHelper(node);
}

int ImageQuadtree::countNumNodesHelper(Node* node) {
```

```
    if (node == nullptr) {
        return 0;
    }

    int count = 1; // Hitung node saat ini
    for (int i = 0; i < 4; i++) {
        count += countNumNodesHelper(node->getChild(i)); // Tambahkan jumlah
node anak
    }

    return count;
}

Node* ImageQuadtree::getRoot() const {
    return root;
}

int ImageQuadtree::getNumNodes() const {
    return numNodes;
}

int ImageQuadtree::getTreeDepth() const {
    return maxDepth;
}
```

4.1.3. Node.h

```
#ifndef NODE_H
#define NODE_H

#include "../utils/RGB.h"
#include <string>
#include <cmath>
#include <vector>

using namespace std;

class Node {
    private:
        bool isLeaf; // true jika node adalah leaf
        int x, y, width, height; // koordinat dan ukuran node
        double errorRate; // error rate node
        RGB avgColor; // nilai rata-rata warna node (RGB)
        Node* children[4]; // array pointer ke Node children

    public:
        // Constructor & Destructor
        Node(bool isLeaf, int x, int y, int width, int height,
vector<vector<RGB>>& imageData);
        ~Node();

        // Metode Pengukuran
        void calculateAverageColor(vector<vector<RGB>>& imageData);
        void calculateErrorRate(std::string methodChoice, vector<vector<RGB>>&
imageData);
        double variance(vector<vector<RGB>>& imageData);
        double meanAbsoluteDeviation(vector<vector<RGB>>& imageData);
        double maxPixelDifference(vector<vector<RGB>>& imageData);
        double entropy(vector<vector<RGB>>& imageData);

        // Getters
        RGB getAvgColor() const;
        int getNodeX() const;
        int getNodeY() const;
        int getNodeWidth() const;
        int getNodeHeight() const;
        Node* getChild(int index);
        double getErrorRate() const;
```

```
        // Setters
        void setChild(int index, Node* child);
        void setAsLeaf(bool value);

        bool isLeafNode() const;
        void deleteChildren();
    };

    #endif
```

4.1.4. Node.cpp

5.

```
/*File: Node.cpp*/
/*Nama: Ahmad Wafi Idzharulhaqq*/

#include "Node.h"
#include <iostream>

using namespace std;

Node::Node(bool isLeaf, int x, int y, int width, int height,
vector<vector<RGB>>& imageData){
    this -> isLeaf = isLeaf;
    this -> x = x;
    this -> y = y;
    this -> width = width;
    this -> height = height;
    this -> errorRate = 0.0;
    calculateAverageColor(imageData);
    for (int i = 0; i < 4; i++){
        children[i] = nullptr;
    }

}
Node::~Node(){
    for (int i = 0; i < 4; i++){
        delete children[i];
    }
}
void Node::calculateAverageColor(vector<vector<RGB>>& imageData){
    long long sumR = 0, sumG = 0, sumB = 0;
    int numPixels = this->width * this->height;
    for (int i = this->y; i < this->y + this->height; ++i) {
        for (int j = this->x; j < this->x + this->width; ++j) {
            sumR += imageData[i][j].r;
            sumG += imageData[i][j].g;
            sumB += imageData[i][j].b;
        }
    }

    this->avgColor.r = (uint8_t)round((double)sumR / numPixels);
    this->avgColor.g = (uint8_t)round((double)sumG / numPixels);
    this->avgColor.b = (uint8_t)round((double)sumB / numPixels);
}

void Node::calculateErrorRate(std::string methodChoice, vector<vector<RGB>>&
imageData){
    if (methodChoice == "1") {
        errorRate = variance(imageData);
    } else if (methodChoice == "2") {
        errorRate = meanAbsoluteDeviation(imageData);
    } else if (methodChoice == "3") {
        errorRate = maxPixelDifference(imageData);
    } else if (methodChoice == "4") {
        errorRate = entropy(imageData);
    }
```

```
}

double Node::variance(vector<vector<RGB>>& imageData){
    double varianceR = 0, varianceG = 0, varianceB = 0;
    int numPixels = width * height;

    for (int i = y; i < y + height; ++i) {
        for (int j = x; j < x + width; ++j) {
            const RGB& pixel = imageData[i][j];
            varianceR += (pixel.r - avgColor.r) * (pixel.r - avgColor.r);
            varianceG += (pixel.g - avgColor.g) * (pixel.g - avgColor.g);
            varianceB += (pixel.b - avgColor.b) * (pixel.b - avgColor.b);
        }
    }

    varianceR /= numPixels;
    varianceG /= numPixels;
    varianceB /= numPixels;

    // Variansi total
    return (varianceR + varianceG + varianceB) / 3.0;
}

double Node::meanAbsoluteDeviation(vector<vector<RGB>>& imageData){
    double madR = 0, madG = 0, madB = 0;
    int numPixels = width * height;

    for (int i = y; i < y + height; ++i) {
        for (int j = x; j < x + width; ++j) {
            const RGB& pixel = imageData[i][j];
            madR += abs((int)pixel.r - (int)avgColor.r);
            madG += abs((int)pixel.g - (int)avgColor.g);
            madB += abs((int)pixel.b - (int)avgColor.b);
        }
    }
    madR /= numPixels;
    madG /= numPixels;
    madB /= numPixels;

    return (madR + madG + madB) / 3.0;
}

double Node::maxPixelDifference(vector<vector<RGB>>& imageData){
    int maxDiffR = 0, maxDiffG = 0, maxDiffB = 0;
    int maxR = 0, maxG = 0, maxB = 0, minR = 255, minG = 255, minB = 255;
    for (int i = y; i < y + height; ++i) {
        for (int j = x; j < x + width; ++j) {
            const RGB& pixel = imageData[i][j];
            maxR = max(maxR, (int)pixel.r);
            maxG = max(maxG, (int)pixel.g);
            maxB = max(maxB, (int)pixel.b);

            minR = min(minR, (int)pixel.r);
            minG = min(minG, (int)pixel.g);
            minB = min(minB, (int)pixel.b);
        }
    }
    maxDiffR = maxR - minR;
    maxDiffG = maxG - minG;
    maxDiffB = maxB - minB;

    return (maxDiffR + maxDiffG + maxDiffB) / 3.0;
}
double Node::entropy(vector<vector<RGB>>& imageData){
    // membuat array frekuensi tiap channel
    int freqR[256] = {0};
    int freqG[256] = {0};
    int freqB[256] = {0};

    int numPixels = width * height;
```

```
    // menghitung frekuensi nilai pixel
    for (int i = y; i < y + height; ++i) {
        for (int j = x; j < x + width; ++j) {
            const RGB& pixel = imageData[i][j];
            freqR[pixel.r]++;
            freqG[pixel.g]++;
            freqB[pixel.b]++;
        }
    }

    // menghitung entropi
    auto calculateEntropy = [&](int freq[256]) -> double {
        double entropy = 0.0;
        for (int i = 0; i < 256; ++i) {
            if (freq[i] > 0) {
                double probability = (double)freq[i] / numPixels;
                entropy -= probability * log2(probability);
            }
        }
        return entropy;
    };

    double entropyR = calculateEntropy(freqR);
    double entropyG = calculateEntropy(freqG);
    double entropyB = calculateEntropy(freqB);

    return (entropyR + entropyG + entropyB) / 3.0;
}

RGB Node::getAvgColor() const{
    return avgColor;
}
int Node::getNodeWidth() const{
    return width;
}
int Node::getNodeHeight() const{
    return height;
}
int Node::getNodeX() const{
    return x;
}
int Node::getNodeY() const{
    return y;
}
Node* Node::getChild(int index){
    return children[index];
}
double Node::getErrorRate() const{
    return errorRate;
}
void Node::setChild(int index, Node* child){
    children[index] = child;
}
bool Node::isLeafNode() const{
    return isLeaf;
}

void Node::setAsLeaf(bool value) {
    this->isLeaf = value;
    for (int i = 0; i < 4; i++) {
        delete children[i];
        children[i] = nullptr;
    }

}

void Node::deleteChildren() {
    for (int i = 0; i < 4; i++) {
        if (children[i] != nullptr) {
```

```
            children[i]->deleteChildren();
            delete children[i];
            children[i] = nullptr;
        }
    }
}
```

5.1.1. ImageLoader.h

```
#ifndef IMAGELOADER_H
#define IMAGELOADER_H

#include "stb_image.h"
#include "RGB.h"
#include <string>
#include <vector>
#include <cstdint>
#include <stdexcept>
#include <iostream>

using namespace std;

class ImageLoader {
private:
    string absPath;
    int width, height;
    vector<vector<RGB>> imageData;

public:
    // Constructor & Destructor
    inline ImageLoader(string absPath);
    inline ~ImageLoader();

    // Load image
    inline void loadImage();

    // Print image information
    inline void printImageInfo() const;

    // Getter for image data
    inline vector<vector<RGB>>& getImageData();

    // Getters for width and height
    inline int getWidth() const;
    inline int getHeight() const;
};

ImageLoader::ImageLoader(string absPath) {
    this->absPath = absPath;
    this->width = 0;
    this->height = 0;
    this->imageData.clear();

    loadImage();
}


ImageLoader::~ImageLoader() {}


void ImageLoader::loadImage() {
    int channels;
    unsigned char* data = stbi_load(this->absPath.c_str(), &this->width,
&this->height, &channels, 3); // Force RGB
    if (!data) {
        throw runtime_error("Image Load Failed: " + absPath);
    }
```

```
    imageData.resize(this->height, vector<RGB>(this->width));

    for (int y = 0; y < this->height; ++y) {
        for (int x = 0; x < this->width; ++x) {
            int idx = (y * this->width + x) * 3;
            imageData[y][x] = {data[idx], data[idx + 1], data[idx + 2]};
        }
    }

    stbi_image_free(data);
}

void ImageLoader::printImageInfo() const {
    cout << "Source: " << absPath << endl;
    cout << "Width: " << width << ", Height: " << height << endl;
}

vector<vector<RGB>>& ImageLoader::getImageData(){
    return imageData;
}

int ImageLoader::getWidth() const {
    return width;
}

int ImageLoader::getHeight() const {
    return height;
}

#endif
```

5.1.2. ImageMaker.h

```
#ifndef IMAGEMAKER_H
#define IMAGEMAKER_H

#include "stb_image_write.h"
#include "RGB.h"
#include <vector>
#include <string>
#include <filesystem>
#include <algorithm>  // for std::transform
#include <cctype>     // for tolower
#include <iostream>

using namespace std;

namespace ImageMaker {
    vector<unsigned char> convertTo1D(const vector<vector<RGB>>& imageData) {
        int height = imageData.size();
        int width = imageData[0].size();
        vector<unsigned char> oneD(width * height * 3);  // RGB = 3 channels

        size_t idx = 0;
        for (int y = 0; y < height; ++y) {
            for (int x = 0; x < width; ++x) {
                const RGB& pixel = imageData[y][x];
                oneD[idx++] = pixel.r;
                oneD[idx++] = pixel.g;
                oneD[idx++] = pixel.b;
            }
        }
        return oneD;
    }
```

```
    void makeImage(const string& outputPath, const vector<vector<RGB>>&
imageData) {
        int height = imageData.size();
        int width = imageData[0].size();
        vector<unsigned char> oneD = convertTo1D(imageData);
        string ext = filesystem::path(outputPath).extension().string();
        std::transform(ext.begin(), ext.end(), ext.begin(), ::tolower);

        bool success = false;
        if (ext == ".jpg" || ext == ".jpeg") {
            // Untuk JPG: gunakan quality lebih rendah karena data sudah
dikompresi oleh quadtree
            success = stbi_write_jpg(outputPath.c_str(), width, height, 3,
oneD.data(), 50);
        } else if (ext == ".png") {
            // Untuk PNG: gunakan filter dan compression level maksimum
            stbi_write_png_compression_level = 9;  // max compression
            success = stbi_write_png(outputPath.c_str(), width, height, 3,
oneD.data(), width * 3);
        }

        if (!success) {
            cerr << "Error: Failed to write image to " << outputPath << endl;
        } else {
            cout << "Image successfully written to " << outputPath << endl;
        }
    }
}

#endif
```

5.1.3. RGB.h

```
#ifndef RGB_H
#define RGB_H

#include <cstdint>

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

#endif
```

5.1.4. main.cpp

```
#include "object/Node.h"
#include "object/ImageQuadtree.h"
#include "utils/ImageLoader.h"
#include "utils/ImageMaker.h"
#include <string>
#include <iostream>
#include <chrono>
#include <filesystem>

using namespace std;
using namespace ImageMaker;

int main() {
    cout << "====Selamat Datang di Image Compressor!!====" << endl;
    cout << "[-] Ketik \"Mulai\" untuk memulai kompresi gambar" << endl;
    cout << "-> ";
```

```
    string inputCommand;
    cin >> inputCommand;
    while (inputCommand != "Mulai") {
        cout << "[-] Ketik \"Mulai\" untuk memulai kompresi gambar" << endl;
        cout << "[-] Ketik \"Batal\" untuk keluar dari program" << endl;
        cout << "--> ";
        cin >> inputCommand;
        if (inputCommand == "Batal") {
            cout << "====Terima kasih sudah menggunakan Image Compressor!====" 
<< endl;
            return 0;
        }
    }

    // Input alamat absolut gambar
    cout << "[-] Masukkan path gambar yang ingin dikompresi: " << endl;
    cout << "--> ";
    string imagePath;
    cin >> imagePath;

    if (!filesystem::exists(imagePath)) {
        cout << "Gambar tidak ditemukan. Silakan periksa path dan coba lagi." 
<< endl;
        return 1;
    }

    // Input metode error measurement yang mau digunakan
    cout << "Metode apa yang akan digunakan untuk error measurement kompresi?" 
<< endl;
    cout << "1. Variance" << endl;
    cout << "2. Mean Absolute Deviation" << endl;
    cout << "3. Maximum Pixel Difference" << endl;
    cout << "4. Entropy" << endl;
    cout << "Masukkan pilihan (1-5): ";
    cin >> inputCommand;
    while (inputCommand < "1" || inputCommand > "5") {
        cout << "[-] Pilihan tidak valid. Silakan pilih antara 1-5." << endl;
        cout << "[-] Ketik \"Batal\" untuk keluar dari program" << endl;
        cout << "--> ";
        cin >> inputCommand;
        if (inputCommand == "Batal") {
            cout << "====Terima kasih sudah menggunakan Image Compressor!====" 
<< endl;
            return 0;
        }
    }

    // Input threshold
    bool validThreshold = false;
    double threshold;
    while (!validThreshold) {
        cout << "Masukkan threshold untuk kompresi: " << endl;
        cin >> threshold;

        if (inputCommand == "1" && (threshold >= 0.0 && threshold <= 1.0)) {
            validThreshold = true;
        } else if (inputCommand == "2" && (threshold >= 0.0 && threshold <= 
255.0)) {
            validThreshold = true;
        } else if (inputCommand == "3" && (threshold >= 0.0 && threshold <= 
255.0)) {
            validThreshold = true;
        } else if (inputCommand == "4" && (threshold >= 0.0 && threshold <= 
8.0)) {
            validThreshold = true;
        } else {
            cout << "Threshold tidak valid untuk metode yang dipilih. Silakan 
masukkan nilai yang sesuai." << endl;
        }
```

```
    }

    // Input ukuran blok minimum
    cout << "Masukkan ukuran minimum blok (pixel): " << endl;
    double minBlockSize;
    cin >> minBlockSize;

    // Input path gambar final
    cout << "Masukkan path untuk menyimpan gambar terkompresi: " << endl;
    string outputPath;
    cin >> outputPath;

    // Menampilkan ukuran awal gambar
    cout << "Memuat gambar..." << endl;
    ImageLoader imageLoader(imagePath);
    imageLoader.loadImage();
    cout << "Gambar berhasil dimuat!" << endl;
    auto originalSize = filesystem::file_size(imagePath);
    cout << "Ukuran gambar sebelum kompresi: " << originalSize << " bytes" <<
endl;

    // Kompresi gambar
    cout << "Memulai kompresi..." << endl;
    auto start = chrono::high_resolution_clock::now();
    ImageQuadtree quadtree(threshold, minBlockSize, imageLoader);
    quadtree.compressImage(quadtree.getRoot(), inputCommand, imageLoader);

    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> elapsed = end - start;

    // Waktu proses
    cout << "Kompresi selesai dalam " << elapsed.count() * 1000 << " ms." <<
endl;

    // Membuat output file dan menampilkan ukuran gambar setelah kompresi
    ImageMaker::makeImage(outputPath, imageLoader.getImageData());
    auto compressedSize = filesystem::file_size(outputPath);
    cout << "Ukuran gambar setelah kompresi: " << compressedSize << " bytes" <<
endl;

    // Menampilkan Persentase kompresi
    double compressionPercent = 100.0 * (1.0 - (double)compressedSize /
originalSize);
    cout << "Persentase kompresi: " << compressionPercent << "%" << endl;

    // Menampilkan Kedalaman pohon
    quadtree.countMaxDepth();
    cout << "Kedalaman pohon: " << quadtree.getTreeDepth() << endl;

    // Menampilkan Banyak simpul pada pohon
    quadtree.countNumNodes(quadtree.getRoot());
    cout << "Banyak simpul pada pohon: " << quadtree.getNumNodes() << endl;

    cout << "Gambar terkompresi berhasil disimpan di: " << outputPath << endl;

    quadtree.~ImageQuadtree();
    cout << "====Terima kasih sudah menggunakan Image Compressor!====" << endl;
    return 0;
}
```

5.2. Pengujian

Proses pengujian dilakukan 2 gambar dengan variasi pengukuran yang berbeda-beda untuk menguji fungsi dan performa program.

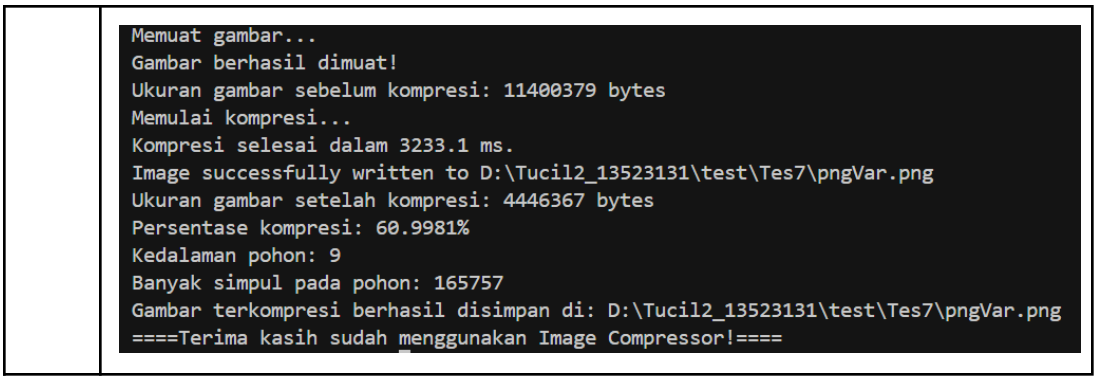
Tabel 2. Hasil Pengujian

No	Hasil Pengujian
A	Input: `====Selamat Datang di Image Compressor!!====` `[-] Ketik "Mulai" untuk memulai kompresi gambar` `-> Mulai` `[-] Masukkan path gambar yang ingin dikompresi:` `--> D:\Tucil2_13523131\test\Tes1\sunraku.jpg` `Metode apa yang akan digunakan untuk error measurement kompresi?` `1. Variance` `2. Mean Absolute Deviation` `3. Maximum Pixel Difference` `4. Entropy` `Masukkan pilihan (1-5): 1` `Masukkan threshold untuk kompresi:` `1` `Masukkan ukuran minimum blok (pixel):` `10` `Masukkan path untuk menyimpan gambar terkompresi:` `D:\Tucil2_13523131\test\Tes1\sunrakuvar.jpg` Output: `Memuat gambar...` `Gambar berhasil dimuat!` `Ukuran gambar sebelum kompresi: 55743 bytes` `Memulai kompresi...` `Kompresi selesai dalam 148.213 ms.` `Image successfully written to D:\Tucil2_13523131\test\Tes1\sunrakuvar.jpg` `Ukuran gambar setelah kompresi: 35122 bytes` `Persentase kompresi: 36.993%` `Kedalaman pohon: 7` `Banyak simpul pada pohon: 15253` `Gambar terkompresi berhasil disimpan di: D:\Tucil2_13523131\test\Tes1\sunrakuvar.jpg` `====Terima kasih sudah menggunakan Image Compressor!====` Input dan Output gambar ada di folder test\Tes1
B	Input & Output: `====Selamat Datang di Image Compressor!!====` `[-] Ketik "Mulai" untuk memulai kompresi gambar` `-> Mulai` `[-] Masukkan path gambar yang ingin dikompresi:` `--> D:\Tucil2_13523131\test\Tes2\FileNotExist.jpg` `Gambar tidak ditemukan. Silakan periksa path dan coba lagi.`

	`` ====Selamat Datang di Image Compressor!!==== [-] Ketik "Mulai" untuk memulai kompresi gambar -> Mulai [-] Masukkan path gambar yang ingin dikompresi: --> D:\Tucil2_13523131\test\Tes1\sunraku.jpg Metode apa yang akan digunakan untuk error measurement kompresi? 1. Variance 2. Mean Absolute Deviation 3. Maximum Pixel Difference 4. Entropy Masukkan pilihan (1-5): 2 Masukkan threshold untuk kompresi: 2 Masukkan ukuran minimum blok (pixel): 10 Masukkan path untuk menyimpan gambar terkompresi: D:\Tucil2_13523131\test\Tes2\FolderNotExist.jpg ```
C	Input: ``` ====Selamat Datang di Image Compressor!!==== [-] Ketik "Mulai" untuk memulai kompresi gambar -> Mulai [-] Masukkan path gambar yang ingin dikompresi: --> D:\Tucil2_13523131\test\Tes3\abstract.jpg Metode apa yang akan digunakan untuk error measurement kompresi? 1. Variance 2. Mean Absolute Deviation 3. Maximum Pixel Difference 4. Entropy Masukkan pilihan (1-5): 2 Masukkan threshold untuk kompresi: 5 Masukkan ukuran minimum blok (pixel): 10 Masukkan path untuk menyimpan gambar terkompresi: D:\Tucil2_13523131\test\Tes3\abstractMAD.jpg ``` ``` Memuat gambar... Gambar berhasil dimuat! Ukuran gambar sebelum kompresi: 341456 bytes Memulai kompresi... Kompresi selesai dalam 340.036 ms. Image successfully written to D:\Tucil2_13523131\test\Tes3\abstractMAD.jpg Ukuran gambar setelah kompresi: 155594 bytes Persentase kompresi: 54.4322% Kedalaman pohon: 7 Banyak simpul pada pohon: 16313 Gambar terkompresi berhasil disimpan di: D:\Tucil2_13523131\test\Tes3\abstractMAD.jpg ====Terima kasih sudah menggunakan Image Compressor!==== ```
D	Input & Output:

	`` ====Selamat Datang di Image Compressor!!==== [-] Ketik "Mulai" untuk memulai kompresi gambar -> Mulai [-] Masukkan path gambar yang ingin dikompresi: --> D:\Tucil2_13523131\test\Tes3\abstract.jpg Metode apa yang akan digunakan untuk error measurement kompresi? 1. Variance 2. Mean Absolute Deviation 3. Maximum Pixel Difference 4. Entropy Masukkan pilihan (1-5): 1 Masukkan threshold untuk kompresi: 100 Threshold tidak valid untuk metode yang dipilih. Silakan masukkan nilai yang sesuai. Masukkan threshold untuk kompresi: ```
E	Input: ``` ====Selamat Datang di Image Compressor!!==== [-] Ketik "Mulai" untuk memulai kompresi gambar -> Mulai [-] Masukkan path gambar yang ingin dikompresi: --> D:\Tucil2_13523131\test\Tes4\human.jpg Metode apa yang akan digunakan untuk error measurement kompresi? 1. Variance 2. Mean Absolute Deviation 3. Maximum Pixel Difference 4. Entropy Masukkan pilihan (1-5): 3 Masukkan threshold untuk kompresi: 2 Masukkan ukuran minimum blok (pixel): 10 Masukkan path untuk menyimpan gambar terkompresi: D:\Tucil2_13523131\test\Tes4\humanMPD.jpg ``` Output: ``` Memuat gambar... Gambar berhasil dimuat! Ukuran gambar sebelum kompresi: 38209 bytes Memulai kompresi... Kompresi selesai dalam 105.826 ms. Image successfully written to D:\Tucil2_13523131\test\Tes4\humanMPD.jpg Ukuran gambar setelah kompresi: 18930 bytes Persentase kompresi: 50.4567% Kedalaman pohon: 6 Banyak simpul pada pohon: 5109 Gambar terkompresi berhasil disimpan di: D:\Tucil2_13523131\test\Tes4\humanMPD.jpg ====Terima kasih sudah menggunakan Image Compressor!==== ```
F	Input:

	`` ====Selamat Datang di Image Compressor!!==== [-] Ketik "Mulai" untuk memulai kompresi gambar -> Mulai [-] Masukkan path gambar yang ingin dikompresi: --> D:\Tucil2_13523131\test\Tes5\colorful.jpg Metode apa yang akan digunakan untuk error measurement kompresi? 1. Variance 2. Mean Absolute Deviation 3. Maximum Pixel Difference 4. Entropy Masukkan pilihan (1-5): 4 Masukkan threshold untuk kompresi: 1 Masukkan ukuran minimum blok (pixel): 1 Masukkan path untuk menyimpan gambar terkompresi: D:\Tucil2_13523131\test\Tes5\colorfulENN.jpg ``` Output: ``` Masukkan path untuk menyimpan gambar terkompresi: D:\Tucil2_13523131\test\Tes5\colorfulENN.jpg Memuat gambar... Gambar berhasil dimuat! Ukuran gambar sebelum kompresi: 202870 bytes Memulai kompresi... Kompresi selesai dalam 1158.95 ms. Image successfully written to D:\Tucil2_13523131\test\Tes5\colorfulENN.jpg Ukuran gambar setelah kompresi: 105921 bytes Persentase kompresi: 47.7887% Kedalaman pohon: 9 Banyak simpul pada pohon: 347985 Gambar terkompresi berhasil disimpan di: D:\Tucil2_13523131\test\Tes5\colorfulENN.jpg ====Terima kasih sudah menggunakan Image Compressor!==== ```
G	``` ====Selamat Datang di Image Compressor!!==== [-] Ketik "Mulai" untuk memulai kompresi gambar -> Mulai [-] Masukkan path gambar yang ingin dikompresi: --> D:\Tucil2_13523131\test\Tes7\png.png Metode apa yang akan digunakan untuk error measurement kompresi? 1. Variance 2. Mean Absolute Deviation 3. Maximum Pixel Difference 4. Entropy Masukkan pilihan (1-5): 1 Masukkan threshold untuk kompresi: 0.9 Masukkan ukuran minimum blok (pixel): 10 Masukkan path untuk menyimpan gambar terkompresi: D:\Tucil2_13523131\test\Tes7\pngVar.png ```



5.3. Hasil Pengujian

Hasil pengujian menunjukkan bahwa penerapan algoritma divide and conquer efektif dalam proses kompresi gambar.

Kompleksitas Waktu. Menggunakan algoritma pada program ini, didapati kompleksitas terburuk adalah $O(n^2 \log n)$ untuk gambar dengan ukuran n x n. Hal ini terjadi karena setiap pembagian blok membentuk pohon quadtree dengan tinggi log(n) dan setiap node memproses pixel sebanyak $O(k^2)$. Namun hal ini masih sangat bergantung pada intensitas elemen yang berada pada gambar, dalam praktiknya.

Kompleksitas Ruang untuk skema terburuk adalah $O(n^2)$, dimana setiap piksel menjadi sebuah node.

6. Kesimpulan

Berdasarkan hasil analisis dan implementasi algoritma kompresi gambar menggunakan pendekatan *divide and conquer* dengan metode Quadtree, dapat disimpulkan bahwa algoritma ini mampu melakukan kompresi secara adaptif dan efisien. Aplikasi dari algoritma *divide and conquer* dan struktur QuadTree memungkinkan pembagian blok yang efisien dan terfokus hanya pada area yang memiliki variasi warna tinggi.

LAMPIRAN

Lampiran 1. Lembar Ketercapaian

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	😎	
2	Program berhasil di jalankan	😎	
3	Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	😎	
4	Mengimplementasi seluruh metode perhitungan error wajib	😎	
5.	Mengerjakan bonus		😭
6.	Program dan laporan dibuat sendiri	😎	

Lampiran 2. Link Github

Tucil2_13523131/test at main · Awfidz/Tucil2_13523131