

Laporan Tugas Kecil 2 IF2211 Strategi Algoritma Kompresi Gambar dengan Metode Quadtree

Muhammad Fathur Rizky
13523105
13523105@std.stei.itb.ac.id

*Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025*

Daftar Isi

1. Permasalahan	0
2. Pendekatan Solusi	1
2.1 Implementasi dalam Kode	1
2.1.a Struktur Data	1
2.1.b Error Measurement Methods	3
a. Variance	3
b. Mean Absolute Deviation (MAD)	4
c. Max Pixel Difference	5
d. Entropy	5
e. Structural Similarity Index Measure (SSIM)	6
2.1.c Membangun Struktur Quadtree	8
2.1.d Menyusun Kembali Representasi Pohon	8
2.2 Analisis Kompleksitas	10
3. Extras	10
GIF Output	10
4. Pengujian Test Case	14
5. Lampiran	18

1. Permasalahan

Kompresi gambar merupakan teknik yang digunakan untuk mengurangi ukuran file gambar dengan tujuan menghemat ruang penyimpanan dan mempercepat proses transmisi data, tanpa mengorbankan kualitas gambar secara signifikan. Permasalahan utama dalam kompresi gambar adalah bagaimana menemukan keseimbangan antara pengurangan ukuran file dan pemeliharaan detail serta kualitas visual. Gambar biasanya mengandung banyak redundansi, terutama pada area dengan warna atau intensitas yang seragam. Namun, di sisi lain, terdapat pula area dengan detail yang kompleks yang harus dipertahankan agar informasi penting tidak hilang. Dengan adanya kebutuhan ini, dibutuhkan metode yang cerdas untuk mengidentifikasi area homogen dan area dengan detail tinggi sehingga kompresi dapat dilakukan secara efisien. Salah satu pendekatan yang populer adalah penggunaan struktur data Quadtree yang dapat membagi gambar menjadi blok-blok berdasarkan keseragaman nilai piksel.

2. Pendekatan Solusi

Pendekatan solusi yang diambil adalah dengan menerapkan algoritma **Divide and Conquer** menggunakan struktur data **Quadtree**. Pada pendekatan ini, gambar diubah menjadi matriks piksel dengan nilai intensitas pada setiap kanal (Red, Green, Blue). Proses kompresi dimulai dengan mengevaluasi blok gambar secara menyeluruh, kemudian menghitung error pada blok menggunakan berbagai metrik seperti Variance, Mean Absolute Deviation (MAD), Max Pixel Difference, Entropy, dan Structural Similarity Index Measure (SSIM). Jika nilai error pada suatu blok melebihi threshold yang telah ditentukan dan ukuran blok cukup besar, blok tersebut akan dibagi menjadi empat sub-blok secara rekursif. Dengan cara ini, area gambar yang homogen dapat direpresentasikan dengan satu nilai rata-rata (normalisasi), sedangkan area yang kompleks—masih memenuhi syarat pembagian—dipertahankan melalui pembagian yang lebih rinci. Pendekatan ini juga memungkinkan penyesuaian dinamis, misalnya dengan mengatur threshold secara otomatis untuk mencapai target persentase kompresi tertentu.

2.1 Implementasi dalam Kode

2.1.a Struktur Data

```
struct Color {
    int r, g, b;
    Color() : r(0), g(0), b(0) {}
    Color(int r, int g, int b) : r(r), g(g), b(b) {}
};
```

Struktur Color merepresentasikan warna dalam format RGB dengan tiga atribut r g dan b yang masing-masing menyimpan nilai intensitas merah hijau dan biru. Konstruktor default Color() menginisialisasi warna hitam (0,0,0) sedangkan konstruktor berparameter Color(int r, int g, int b)

memungkinkan inisialisasi warna dengan nilai yang ditentukan. Struktur ini digunakan untuk menyimpan dan memproses warna dalam Quadtree saat melakukan kompresi gambar.

```
class QuadtreeNode {
public:
    int x, y, width, height, depth;
    bool isLeaf;
    Color color;
    QuadtreeNode* children[4];

    QuadtreeNode(int _x, int _y, int _width, int _height);
    ~QuadtreeNode();
};
```

Kode di atas mendefinisikan kelas QuadtreeNode yang merepresentasikan simpul dalam struktur Quadtree. Setiap simpul dalam Quadtree menyimpan informasi mengenai posisi dan ukuran blok gambar yang diwakilinya melalui atribut x y width dan height. Selain itu atribut depth digunakan untuk menyimpan kedalaman simpul dalam pohon yang membantu dalam proses rekursif pembentukan Quadtree.

Simpul dapat berupa daun (leaf node) atau bukan yang ditentukan oleh atribut isLeaf. Jika isLeaf == true maka simpul tersebut tidak memiliki anak dan blok yang direpresentasikan dianggap cukup homogen untuk tidak dibagi lebih lanjut. Setiap simpul juga menyimpan warna rata-rata dari blok yang diwakilinya dalam atribut color yang digunakan untuk menentukan warna representasi pada gambar hasil kompresi.

Selain itu QuadtreeNode memiliki empat anak (children) yang direpresentasikan sebagai array children[4] yang masing-masing menunjuk ke sub-blok dalam Quadtree. Hal ini sesuai dengan sifat dasar Quadtree yang membagi setiap blok menjadi empat bagian lebih kecil ketika nilai error melebihi ambang batas tertentu.

```
class Quadtree {
private:
    std::vector<std::vector<Color>> pixelData;
    QuadtreeNode* root;
    double threshold;
    int minBlockSize;
    Metric* metric;
    QuadtreeNode* buildQuadtree(int x, int y, int width, int height,
                                int depth = 0);
    Color calculateAverageColor(const std::vector<std::vector<Color>>& data,
                               int x, int y, int width, int height);

public:
    Quadtree(const std::vector<std::vector<Color>>& data, double thresh,
             Metric* metric, int minBlockSize);
    ~Quadtree();
    int getTreeDepth() const;
    int getNodeCount() const;
```

```

int getLeafCount() const;
FIBITMAP* createImage(int customDepth, bool showLines);
QuadtreeNode* getRoot() const { return root; }
};

```

Kelas Quadtree merepresentasikan struktur Quadtree yang digunakan untuk kompresi gambar dengan algoritma Divide and Conquer. Kelas ini menyimpan data piksel dalam pixelData yang digunakan untuk membangun Quadtree secara rekursif dengan buildQuadtree(). Proses pembagian blok didasarkan pada threshold error dan ukuran minimum blok (minBlockSize) dengan bantuan metrik yang diterapkan melalui objek metric.

Konstruktor Quadtree() menginisialisasi Quadtree dengan data gambar sedangkan destruktur ~Quadtree() bertanggung jawab untuk membersihkan memori. Fungsi calculateAverageColor() menghitung warna rata-rata suatu blok yang digunakan dalam simpul Quadtree.

Metode getTreeDepth() getNodeCount() dan getLeafCount() digunakan untuk mendapatkan informasi tentang struktur pohon. Fungsi createImage() menghasilkan gambar berdasarkan struktur Quadtree. Atribut root menyimpan simpul akar Quadtree yang dapat diakses melalui getRoot().

2.1.b Error Measurement Methods

a. Variance

```

double VarianceMetric::compute(const std::vector<std::vector<Color>>& pixels,
                                int x, int y, int width, int height) {
    long sumR = 0, sumG = 0, sumB = 0;
    int count = width * height;
    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            sumR += pixels[i][j].r;
            sumG += pixels[i][j].g;
            sumB += pixels[i][j].b;
        }
    }
    double avgR = sumR / (double)count;
    double avgG = sumG / (double)count;
    double avgB = sumB / (double)count;
    double variance = 0.0;
    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            variance += (pixels[i][j].r - avgR) * (pixels[i][j].r - avgR) +
                        (pixels[i][j].g - avgG) * (pixels[i][j].g - avgG) +
                        (pixels[i][j].b - avgB) * (pixels[i][j].b - avgB);
        }
    }
    return variance / count;
}

```

Variance adalah ukuran statistik yang digunakan untuk mengukur seberapa besar penyebaran nilai intensitas warna dalam suatu blok gambar terhadap rata-ratanya. Dalam konteks kompresi citra variance membantu mengidentifikasi apakah suatu blok homogen atau memiliki detail kompleks. Nilai variance tinggi menunjukkan variasi warna besar sedangkan nilai rendah menunjukkan keseragaman.

Kelebihan variance yaitu perhitungannya sederhana dan efektif dalam mendeteksi kompleksitas blok sehingga cocok digunakan dalam algoritma Quadtree. Namun kekurangannya adalah tidak mempertimbangkan struktur visual gambar serta sensitif terhadap pencahayaan dan perbedaan warna yang tidak selalu bermakna secara visual.

b. Mean Absolute Deviation (MAD)

```
double MADMetric::compute(const std::vector<std::vector<Color>>& pixels, int
x,
                           int y, int width, int height) {
    long sumR = 0, sumG = 0, sumB = 0;
    int count = width * height;
    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            sumR += pixels[i][j].r;
            sumG += pixels[i][j].g;
            sumB += pixels[i][j].b;
        }
    }
    double avgR = sumR / (double)count;
    double avgG = sumG / (double)count;
    double avgB = sumB / (double)count;
    double mad = 0.0;
    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            mad += std::abs(pixels[i][j].r - avgR) + std::abs(pixels[i][j].g -
avgG) + std::abs(pixels[i][j].b - avgB);
        }
    }
    return mad / count;
}
```

Mean Absolute Deviation (MAD) adalah metrik yang mengukur seberapa jauh nilai piksel dalam suatu blok gambar menyimpang dari rata-ratanya. Fungsi MADMetric::compute pertama-tama menghitung rata-rata intensitas warna untuk setiap kanal (R, G, B) dalam blok gambar lalu menghitung selisih absolut antara setiap piksel dan nilai rata-rata kemudian menjumlahkan hasilnya. Nilai MAD yang lebih tinggi menunjukkan blok dengan variasi warna yang lebih besar sedangkan nilai yang lebih rendah menunjukkan area yang lebih seragam. Metode ini lebih robust terhadap outlier dibanding variance karena hanya menggunakan perbedaan absolut tanpa mengkuadratkan nilai deviasi sehingga lebih sesuai untuk mendeteksi perubahan warna tanpa terpengaruh nilai ekstrem.

c. Max Pixel Difference

```
double MaxPixelDifferenceMetric::compute(
    const std::vector<std::vector<Color>>& pixels, int x, int y, int width,
    int height) {
    int minR = 255, minG = 255, minB = 255;
    int maxR = 0, maxG = 0, maxB = 0;
    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            const Color& c = pixels[i][j];
            minR = std::min(minR, c.r);
            minG = std::min(minG, c.g);
            minB = std::min(minB, c.b);
            maxR = std::max(maxR, c.r);
            maxG = std::max(maxG, c.g);
            maxB = std::max(maxB, c.b);
        }
    }
    double dR = maxR - minR;
    double dG = maxG - minG;
    double dB = maxB - minB;
    return (dR + dG + dB) / 3.0;
}
```

Metrik *Max Pixel Difference* menghitung selisih antara nilai piksel maksimum dan minimum pada setiap kanal warna (R, G, B) dalam suatu blok gambar. Nilai selisih ini mencerminkan rentang variasi warna di blok tersebut. Jika rentangnya besar maka blok dianggap memiliki banyak perbedaan warna sedangkan jika kecil blok dianggap homogen. Metrik ini sangat sederhana dan cepat dihitung namun kurang sensitif terhadap variasi halus tidak mempertimbangkan distribusi nilai piksel secara keseluruhan dan rentan terhadap outlier di mana satu piksel dengan nilai ekstrem dapat secara signifikan memengaruhi hasil perhitungan.

d. Entropy

```
double EntropyMetric::compute(const std::vector<std::vector<Color>>& pixels,
                               int x, int y, int width, int height) {
    int count = width * height;
    std::vector<int> histR(256, 0), histG(256, 0), histB(256, 0);
    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            const Color& c = pixels[i][j];
            histR[c.r]++;
            histG[c.g]++;
            histB[c.b]++;
        }
    }
    double entropyR = 0.0, entropyG = 0.0, entropyB = 0.0;
    for (int i = 0; i < 256; i++) {
        if (histR[i] > 0) {
            double p = histR[i] / (double)count;
            entropyR -= p * std::log2(p);
            entropyG -= p * std::log2(p);
            entropyB -= p * std::log2(p);
        }
    }
    return (entropyR + entropyG + entropyB) / 3.0;
}
```

```

    }
    if (histG[i] > 0) {
        double p = histG[i] / (double)count;
        entropyG -= p * std::log2(p);
    }
    if (histB[i] > 0) {
        double p = histB[i] / (double)count;
        entropyB -= p * std::log2(p);
    }
}
return (entropyR + entropyG + entropyB) / 3.0;
}

```

Metrik Entropy mengukur ketidakteraturan informasi dalam suatu blok gambar berdasarkan distribusi intensitas warna yang mencerminkan tingkat ketidakpastian atau variasi warna dalam blok tersebut. Fungsi ini menghitung histogram untuk tiap kanal warna (R, G, B) kemudian menghitung entropi menggunakan rumus $-p \log_2 p$ untuk setiap tingkat intensitas yang muncul.

Nilai entropi tinggi menunjukkan variasi warna yang besar dan keberagaman pola sedangkan nilai rendah menunjukkan keseragaman warna yang lebih dominan. Metrik ini sangat efektif dalam menangkap detail tekstur dan pola dalam gambar namun lebih kompleks secara komputasi dibanding metrik lainnya serta sensitif terhadap distribusi lokal yang tidak selalu tampak secara visual.

e. Structural Similarity Index Measure (SSIM)

```

SSIMetric::SSIMetric(double threshold) : threshold(threshold) {}

double SSIMetric::compute(const std::vector<std::vector<Color>>& pixels, int
x,
                           int y, int width, int height) {
const double C1 = 0.01 * 255 * 0.01 * 255; // (K1*L)^2
const double C2 = 0.03 * 255 * 0.03 * 255; // (K2*L)^2

double mean_original = 0.0;
for (int i = y; i < y + height; i++) {
    for (int j = x; j < x + width; j++) {
        double intensity = 0.299 * pixels[i][j].r + 0.587 * pixels[i][j].g +
                           0.114 * pixels[i][j].b;
        mean_original += intensity;
    }
}

int num_pixels = width * height;
mean_original /= num_pixels;

double mean_x = mean_original;
double mean_y = mean_original;

double var_x = 0.0;
for (int i = y; i < y + height; i++) {

```

```

        for (int j = x; j < x + width; j++) {
            double intensity_x = 0.299 * pixels[i][j].r + 0.587 * pixels[i][j].g +
                0.114 * pixels[i][j].b;
            var_x += (intensity_x - mean_x) * (intensity_x - mean_x);
        }
    }
    double var_y = 0.0;
    double covar = 0.0;

    var_x /= (num_pixels - 1);

    double ssim =
        (2 * mean_x * mean_y + C1) * (2 * covar + C2) /
        ((mean_x * mean_x + mean_y * mean_y + C1) * (var_x + var_y + C2));

    return 2 * threshold - ssim;
}

```

Kode di atas mengimplementasikan metode untuk menghitung Structural Similarity Index Measure (SSIM) pada sebuah blok citra. Fungsi compute menerima parameter berupa matriks dua dimensi dari objek Color yang mewakili citra, beserta koordinat (x, y) dan dimensi (width, height) dari blok yang akan dievaluasi. Dalam konteks ini, diasumsikan bahwa citra asli (disebut sebagai "y") telah di-replace dengan satu warna monoton, sehingga seluruh area blok tersebut memiliki nilai intensitas yang sama.

Pertama-tama, kode menghitung rata-rata intensitas (mean) pada blok citra asli menggunakan koefisien 0.299, 0.587, dan 0.114, yang merupakan bobot untuk channel merah, hijau, dan biru dalam menentukan intensitas. Nilai rata-rata yang dihitung, mean_original, kemudian digunakan untuk menetapkan nilai mean_x dan mean_y. Dengan asumsi bahwa citra asli sudah homogen (monoton), maka perbedaan nilai antar piksel sangat kecil atau bahkan nol.

Variansi untuk citra "x" dihitung berdasarkan perbedaan intensitas tiap piksel dari mean_x. Namun, karena citra "y" diasumsikan sebagai warna monoton, variansi var_y dan kovarians covar dibiarkan nol, yang menyederhanakan perhitungan SSIM.

Konstanta C1 dan C2 didefinisikan untuk menjaga kestabilan perhitungan numerik. Penggunaan konstanta ini merupakan praktik standar dalam perhitungan SSIM, yang bertujuan untuk menghindari pembagian dengan nol dan mengurangi sensitivitas perhitungan terhadap noise atau fluktuasi kecil pada nilai rata-rata dan variansi. Nilai-nilai C1 dan C2 dihitung berdasarkan nilai maksimum intensitas ($L = 255$), sehingga memberikan batas bawah yang aman dalam perhitungan.

Selanjutnya, SSIM dihitung dengan menggunakan formula standar, yang melibatkan nilai mean, variansi, dan kovarians. Hasil SSIM tersebut kemudian diubah melalui transformasi $2 * \text{threshold} - \text{ssim}$ yang bertujuan agar memanipulasi bentuk $\text{var} \geq \text{threshold}$ menjadi $\text{var} \leq \text{threshold}$.

2.1.c Membangun Struktur Quadtree

```
QuadtreeNode* Quadtree::buildQuadtree(int x, int y, int width, int height,
                                       int depth) {
    float var = metric->compute(pixelData, x, y, width, height);
    QuadtreeNode* node = new QuadtreeNode(x, y, width, height);
    node->depth = depth;
    node->color = calculateAverageColor(pixelData, x, y, width, height);
    node->isLeaf = true;
    if (var >= threshold && (width / 2) * (height / 2) >= minBlockSize) {
        node->isLeaf = false;
        int halfWidth = width / 2;
        int halfHeight = height / 2;
        node->children[0] = buildQuadtree(x, y, halfWidth, halfHeight, depth + 1);
        node->children[1] = buildQuadtree(x + halfWidth, y, width - halfWidth,
                                         halfHeight, depth + 1);
        node->children[2] = buildQuadtree(x, y + halfHeight, halfWidth,
                                         height - halfHeight, depth + 1);
        node->children[3] =
            buildQuadtree(x + halfWidth, y + halfHeight, width - halfWidth,
                          height - halfHeight, depth + 1);
    }
    return node;
}
```

Fungsi buildQuadtree digunakan untuk membangun struktur Quadtree secara rekursif dari blok-blok gambar berdasarkan nilai variansi yang dihitung menggunakan metrik tertentu. Fungsi ini menerima koordinat awal blok yaitu x dan y serta lebar dan tinggi blok dan juga kedalaman simpul saat ini. Pertama fungsi menghitung nilai error atau variasi warna dalam blok menggunakan fungsi metric->compute kemudian membuat simpul Quadtree baru dengan informasi posisi ukuran dan warna rata-rata dari blok tersebut. Secara default simpul dianggap sebagai daun. Jika nilai error melebihi threshold yang telah ditentukan dan ukuran blok hasil pembagian masih lebih besar atau sama dengan ukuran minimum maka blok akan dibagi menjadi empat bagian dan fungsi buildQuadtree dipanggil kembali secara rekursif untuk keempat anaknya. Dengan cara ini, pohon Quadtree akan terbentuk dari akar hingga ke daun di mana pembagian blok hanya dilakukan jika memang memenuhi persyaratan nilai threshold. Fungsi ini akan mengembalikan pointer ke simpul yang telah dibangun dan digunakan untuk membentuk keseluruhan struktur pohon.

2.1.d Menyusun Kembali Representasi Pohon

```
std::function<void(QuadtreeNode*)> drawNode = [&] (QuadtreeNode* node) {
    if (node->isLeaf) {
        for (int y = node->y; y < node->y + node->height; y++) {
            for (int x = node->x; x < node->x + node->width; x++) {
                RGBQUAD col;
                col.rgbRed = node->color.r;
                col.rgbGreen = node->color.g;
                col.rgbBlue = node->color.b;
```

```

        FreeImage_SetPixelColor(bitmap, x, y, &col);
    }
}

if (drawOutline) {
    RGBQUAD outline = {0, 0, 0, 0};
    for (int x = node->x; x < node->x + node->width; x++)
        FreeImage_SetPixelColor(bitmap, x, node->y, &outline);
    for (int x = node->x; x < node->x + node->width; x++)
        FreeImage_SetPixelColor(bitmap, x, node->y + node->height - 1,
                               &outline);
    for (int y = node->y; y < node->y + node->height; y++)
        FreeImage_SetPixelColor(bitmap, node->x, y, &outline);
    for (int y = node->y; y < node->y + node->height; y++)
        FreeImage_SetPixelColor(bitmap, node->x + node->width - 1, y,
                               &outline);
}
} else {
    for (int i = 0; i < 4; i++) {
        if (node->children[i] != nullptr) drawNode(node->children[i]);
    }
}
};


```

Fungsi drawNode merupakan sebuah lambda function bertipe std::function yang digunakan untuk menggambar isi pohon Quadtree ke dalam sebuah bitmap menggunakan pustaka FreeImage. Fungsi ini menerima sebuah simpul QuadtreeNode sebagai parameter dan akan menggambar seluruh area yang diwakili oleh simpul tersebut sesuai dengan warna rata-rata yang disimpan di dalamnya.

Jika simpul merupakan daun maka fungsi akan mengisi setiap piksel dalam area simpul dengan warna RGB sesuai dengan nilai node->color menggunakan FreeImage_SetPixelColor. Selain itu jika opsi drawOutline bernilai true maka fungsi juga menggambar garis tepi hitam di sekeliling blok dengan mengganti warna piksel di baris atas baris bawah kolom kiri dan kolom kanan dari area simpul tersebut.

Sebaliknya jika simpul bukan daun maka fungsi akan memanggil dirinya sendiri secara rekursif untuk keempat anak dari simpul tersebut asalkan anak tersebut tidak bernilai null. Dengan pendekatan rekursif ini keseluruhan struktur pohon Quadtree dapat divisualisasikan ke dalam gambar akhir di mana setiap simpul daun akan ditampilkan sebagai blok warna solid yang sesuai dan terpisah oleh garis batas jika diaktifkan. Fungsi ini sangat penting dalam proses visualisasi hasil kompresi gambar karena secara langsung membentuk tampilan akhir dari gambar yang telah diproses oleh algoritma Quadtree.

2.2 Analisis Kompleksitas

Dalam algoritma kompresi gambar berbasis Quadtree, proses utama yang memengaruhi kompleksitas adalah pembentukan pohon melalui pembagian blok gambar secara rekursif. Analisis kompleksitas dilakukan dengan melihat bagaimana fungsi buildQuadtree memecah blok gambar berdasarkan nilai error (seperti variance, MAD, dan lainnya) hingga mencapai batas threshold atau ukuran blok minimum.

Misalkan ukuran gambar adalah $n \times n$ piksel. Setiap kali algoritma membagi suatu blok, blok tersebut dipisah menjadi 4 sub-blok berukuran setengah dari lebar dan tinggi sebelumnya. Jika pembagian ini dilakukan sampai ke tingkat kedalaman maksimum $\log_2 n$ (karena ukuran blok berkurang setengah setiap level), maka jumlah maksimum blok yang bisa terbentuk adalah:

$$T(n) = 4^0 + 4^1 + 4^2 + \dots + 4^{\log_2 n}$$

Ini adalah deret geometri dengan rasio 4, sehingga jumlah total simpul (node) dalam kasus terburuk menjadi

$$T(n) = (4^{(\log_2 n + 1)} - 1) / (4 - 1) \approx O(n^2)$$

Hal ini terjadi karena $4^{\log_2 n} = n^2$. Maka, meskipun jumlah level maksimum adalah $\log_2 n$, jumlah total simpul atau blok bisa mencapai $O(n^2)$ pada kasus terburuk di mana semua blok terus dibagi sampai ukuran terkecil. Selanjutnya, dalam setiap pemanggilan compute pada metrik (misalnya variance, MAD, entropy), seluruh piksel dalam blok tersebut dihitung satu per satu. Karena setiap piksel hanya dihitung satu kali untuk satu blok dan semua blok totalnya $O(n^2)$, maka secara keseluruhan proses evaluasi metrik juga memerlukan $O(n^2)$ operasi.

Dengan demikian, kompleksitas waktu total algoritma adalah $T(n) = O(n^2)$, baik dari segi pembentukan pohon maupun evaluasi metrik pada tiap blok. Pada praktiknya kompleksitas bisa lebih rendah jika banyak blok dianggap homogen sejak awal dan tidak dibagi lebih lanjut sehingga proses rekursif berhenti lebih awal.

3. Extras

GIF Output

```
void ImageCompressor::saveGif() {
    int maxDepth = quadtree->getTreeDepth();
    std::vector<std::string> frameFilenames;
    auto isFileExist = [&] (const std::string& filename) -> bool {
```

```

    struct stat buffer;
    return (stat(filename.c_str(), &buffer) == 0);
}

for (int depth = 0; depth <= maxDepth; depth++) {
    FIBITMAP* frameBitmap = quadtree->createImage(depth, true);
    if (!frameBitmap) {
        std::cerr << "Error: Failed to create image for depth " << depth
            << std::endl;
        continue;
    }
    std::ostringstream oss;
    oss << "frame_" << depth << ".png";
    std::string frameFile = oss.str();
    frameFilenames.push_back(frameFile);

    if (!FreeImage_Save(FIF_PNG, frameBitmap, frameFile.c_str(), 0)) {
        std::cerr << "Error: Failed to save frame " << depth << std::endl;
    }
    FreeImage_Unload(frameBitmap);
}

std::ostringstream cmd;
cmd << "convert -delay 50 -loop 0";
for (const auto& frame : frameFilenames) {
    if (isFileExist(frame))
        cmd << " " << frame;
    else
        std::cerr << "Warning: Frame file " << frame << " does not exist."
            << std::endl;
}
cmd << " " << gifPath;

int ret = system(cmd.str().c_str());
if (ret != 0) {
    std::cerr << "Error: Failed to create GIF using ImageMagick. Command: "
        << cmd.str() << std::endl;
}

for (const auto& frame : frameFilenames) {
    if (isFileExist(frame)) {
        if (std::remove(frame.c_str()) != 0)
            std::cerr << "Error: Failed to remove temporary frame " << frame
                << std::endl;
    }
}
std::cout << "[OUTPUT] GIF saved at: " << gifPath << std::endl;
}

```

Fungsi Quadtree::createlimage bertujuan untuk menghasilkan sebuah gambar (bitmap) berdasarkan struktur quadtree yang telah dibangun. Proses dimulai dengan menentukan lebar dan tinggi gambar dari data piksel yang tersimpan, di mana lebar diambil dari jumlah kolom dan tinggi dari jumlah baris. Setelah itu, library FreeImage diinisialisasi dan bitmap dialokasikan dengan kedalaman warna 24-bit. Jika alokasi gagal, fungsi akan mencetak pesan kesalahan,

menonaktifkan FreeImage, dan mengembalikan nullptr. Selanjutnya, fungsi mendefinisikan sebuah lambda rekursif untuk menelusuri struktur quadtree mulai dari root. Setiap node yang ditemui akan diperiksa; jika node tersebut merupakan daun atau kedalamannya sudah mencapai atau melebihi customDepth (maxDepth), maka area yang diwakili oleh node tersebut diwarnai dengan warna yang tersimpan di dalam node. Proses pengisian warna dilakukan dengan iterasi pada setiap piksel dalam area tersebut dan mengatur nilai warna dengan fungsi FreeImage_SetPixelColor. Jika node bukan daun dan kedalamannya masih di bawah customDepth, maka fungsi ini akan memanggil dirinya sendiri untuk setiap anak node yang ada. Setelah seluruh struktur quadtree telah diproses dan digambar ke dalam bitmap, library FreeImage dinonaktifkan dan pointer ke bitmap dikembalikan. Meskipun terdapat parameter showLines, parameter tersebut tidak digunakan dalam implementasi saat ini, yang mengindikasikan bahwa fitur untuk menampilkan garis batas antar node mungkin akan berguna di masa depan.

Secara keseluruhan, fungsi ini mengonversi struktur hierarkis quadtree menjadi representasi visual yang menunjukkan pembagian wilayah gambar berdasarkan informasi warna yang tersimpan di setiap node.

```

void ImageCompressor::saveGif() {
    int maxDepth = quadtree->getTreeDepth();
    std::vector<std::string> frameFilenames;
    auto isFileExist = [&](const std::string& filename) -> bool {
        struct stat buffer;
        return (stat(filename.c_str(), &buffer) == 0);
    };

    for (int depth = 0; depth <= maxDepth; depth++) {
        FIBITMAP* frameBitmap = quadtree->createImage(depth, true);
        if (!frameBitmap) {
            std::cerr << "Error: Failed to create image for depth " << depth
                << std::endl;
            continue;
        }
        std::ostringstream oss;
        oss << "frame_" << depth << ".png";
        std::string frameFile = oss.str();
        frameFilenames.push_back(frameFile);

        if (!FreeImage_Save(FIF_PNG, frameBitmap, frameFile.c_str(), 0)) {
            std::cerr << "Error: Failed to save frame " << depth << std::endl;
        }
        FreeImage_Unload(frameBitmap);
    }

    std::ostringstream cmd;
    cmd << "convert -delay 50 -loop 0";
    for (const auto& frame : frameFilenames) {
        if (isFileExist(frame))
            cmd << " " << frame;
        else
    }
}

```

```

        std::cerr << "Warning: Frame file " << frame << " does not exist."
        << std::endl;
    }
cmd << " " << gifPath;

int ret = system(cmd.str().c_str());
if (ret != 0) {
    std::cerr << "Error: Failed to create GIF using ImageMagick. Command: "
    << cmd.str() << std::endl;
}

for (const auto& frame : frameFilenames) {
    if (isFileExist(frame)) {
        if (std::remove(frame.c_str()) != 0)
            std::cerr << "Error: Failed to remove temporary frame " << frame
            << std::endl;
    }
}
std::cout << "[OUTPUT] GIF saved at: " << gifPath << std::endl;
}

```

Fungsi saveGif bertujuan untuk menghasilkan dan menyimpan sebuah animasi GIF yang menggambarkan proses kompresi citra berdasarkan struktur quadtree. Pertama, fungsi mengambil kedalaman maksimum dari quadtree dengan memanggil getTreeDepth dan menyimpan nilai tersebut dalam variabel maxDepth. Selanjutnya, sebuah vector digunakan untuk menyimpan nama-nama file frame yang akan dihasilkan, serta sebuah fungsi lambda isFileExist didefinisikan untuk mengecek keberadaan file dengan menggunakan fungsi stat.

Kemudian, fungsi melakukan iterasi dari kedalaman 0 hingga maxDepth. Pada setiap iterasi, fungsi memanggil createImage pada quadtree dengan parameter depth saat ini dan nilai true untuk opsi showLines, yang menghasilkan sebuah bitmap (frame) dari citra pada tingkat kedalaman tersebut. Jika bitmap berhasil dibuat, nama file frame dibuat dengan format "frame_depth.png" dan nama tersebut disimpan dalam vector. Gambar frame kemudian disimpan ke dalam file dengan menggunakan FreeImage_Save, dan setelah itu bitmap dilepaskan dari memori melalui FreeImage_Unload. Jika terjadi kegagalan dalam pembuatan atau penyimpanan frame, fungsi mencetak pesan kesalahan ke standard error.

Setelah seluruh frame dibuat dan disimpan, fungsi menyusun sebuah perintah command-line untuk menggabungkan frame-frame tersebut menjadi sebuah file GIF. Perintah tersebut diawali dengan "convert -delay 50 -loop 0", diikuti oleh daftar file frame yang telah diverifikasi keberadaannya dengan fungsi isFileExist, dan diakhiri dengan path output GIF yang tersimpan dalam variabel gifPath. Perintah ini kemudian dijalankan menggunakan fungsi system. Jika perintah tersebut gagal (ditandai dengan nilai kembalian non-nol), maka fungsi akan mencetak pesan error yang menunjukkan perintah yang gagal dijalankan.

Akhirnya, setelah proses pembuatan GIF selesai, fungsi menghapus file-file frame sementara dengan memeriksa keberadaan setiap file dan menggunakan std::remove untuk menghapusnya. Setelah semua file frame berhasil dihapus, fungsi mencetak pesan output ke

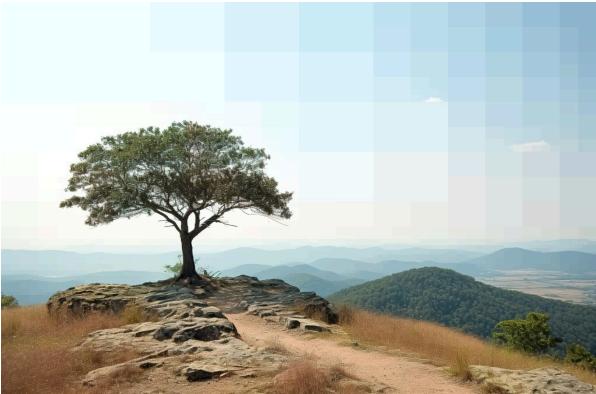
layar yang menunjukkan bahwa GIF telah berhasil disimpan di lokasi yang ditentukan. Dengan demikian, fungsi saveGif menggabungkan proses pembuatan frame pada tiap tingkat kedalaman quadtree, penyimpanan frame sebagai file sementara, penggabungan file-file tersebut menjadi sebuah animasi GIF, dan pembersihan file sementara setelah proses selesai.

4. Pengujian Test Case

Test Case #0 (Example)

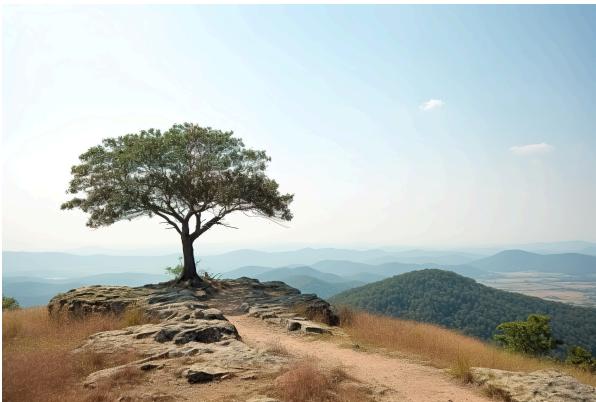
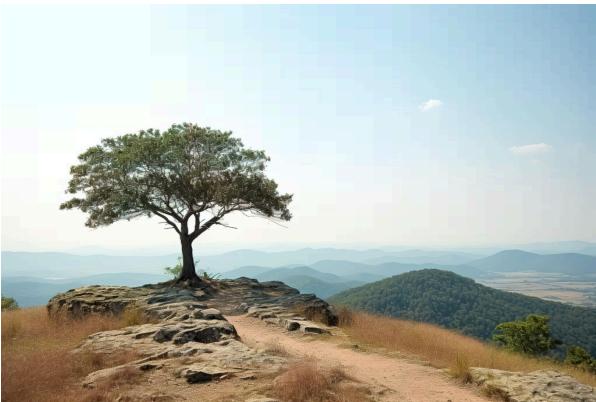
Input	Output
path/to/input.[jpg,png,jpeg] errorMethod threshold minBlockSize compressionTarget path/to/output.[jpg,png,jpeg] path/to/output.gif test-alone.jpg 5 0.8 10 0 test-alone-compressed.png test-alone.gif	----- [STATS] ----- [OUTPUT] Compressed image saved at: [OUTPUT] GIF saved at: [INFO] Max Depth: [INFO] Execution Time: [INFO] Original File Size: [INFO] Compressed File Size: [INFO] Compression Percentage:

Test Case #1 (Positive: SSIM)

Input	Output
 test-alone.jpg 5	 ----- [STATS] ----- [OUTPUT] Compressed image saved at:

<pre>0.8 10 0 test/alone-compressed.png test/alone.gif</pre>	<pre>test/alone-compressed.png [OUTPUT] GIF saved at: test/alone.gif [INFO] Max Depth: 11 [INFO] Execution Time: 3.40 sec [INFO] Original File Size: 3.51 MB [INFO] Compressed File Size: 1.58 MB [INFO] Compression Percentage: 54.86%</pre>
--	---

Test Case #2 (Positive: MAD)

Input	Output
 <pre>test/alone.jpg 2 3 10 0 test/alone-compressed.png test/alone.gif</pre>	 <pre>----- [STATS] ----- [OUTPUT] Compressed image saved at: test/alone-compressed.png [OUTPUT] GIF saved at: test/alone.gif [INFO] Max Depth: 11 [INFO] Execution Time: 4.49 sec [INFO] Original File Size: 3.51 MB [INFO] Compressed File Size: 1.60 MB [INFO] Compression Percentage: 54.40%</pre>

Test Case #3 (Positive: Max Pixel Difference)

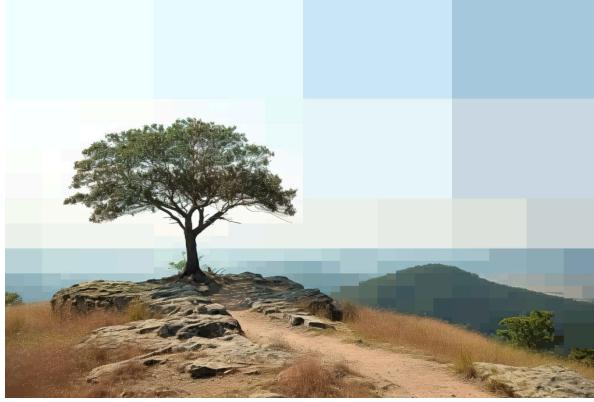
Input	Output
-------	--------

 test/alone.jpg 3 20 10 0 test/alone-compressed.png test/alone.gif	 ----- [STATS] ----- [OUTPUT] Compressed image saved at: test/alone-compressed.png [OUTPUT] GIF saved at: test/alone.gif [INFO] Max Depth: 11 [INFO] Execution Time: 2.76 sec [INFO] Original File Size: 3.51 MB [INFO] Compressed File Size: 1.58 MB [INFO] Compression Percentage: 54.81%
--	--

Test Case #4 (Positive: Entropy)

Input	Output
 test/alone.jpg 4 2 10 0 test/alone-compressed.png test/alone.gif	 ----- [STATS] ----- [OUTPUT] Compressed image saved at: test/alone-compressed.png [OUTPUT] GIF saved at: test/alone.gif [INFO] Max Depth: 11 [INFO] Execution Time: 4.29 sec [INFO] Original File Size: 3.51 MB [INFO] Compressed File Size: 1.60 MB [INFO] Compression Percentage: 54.42%

Test Case #5 (Positive: Variance)

Input	Output
 test/alone.jpg 1 1000 10 0 test/alone-compressed.png test/alone.gif	 ----- [STATS] ----- [OUTPUT] Compressed image saved at: test/alone-compressed.png [OUTPUT] GIF saved at: test/alone.gif [INFO] Max Depth: 11 [INFO] Execution Time: 3.91 sec [INFO] Original File Size: 3.51 MB [INFO] Compressed File Size: 1.42 MB [INFO] Compression Percentage: 59.40%

Test Case #6 (Negative: File Input Not Found)

Input	Output
not_found.jpg 1 1000 10 0 test/alone-compressed.png test/alone.gif	Error: Cannot load image from not_found.jpg make: *** [Makefile:13: all] Error 1

Test Case #7 (Negative: minBlockSize < 1)

Input	Output
-------	--------

<pre>test/alone.jpg 1 10 0 1 test/alone-compressed.png test/alone.gif</pre>	<p>Error: Minimum block size must be greater than 0! make: *** [Makefile:13: all] Error 1</p>
--	---

Test Case #8 (Negative: Error Method < 1 or Error Method > 5)

Input	Output
<pre>test/alone.jpg 6 10 10 1 test/alone-compressed.png test/alone.gif test/alone.jpg 0 10 10 1 test/alone-compressed.png test/alone.gif</pre>	<p>Error: Invalid error method! make: *** [Makefile:13: all] Error 1</p>

5. Lampiran

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	<input checked="" type="checkbox"/>	
8. Program dan laporan dibuat (kelompok) sendiri	<input checked="" type="checkbox"/>	

Tautan Repository: https://github.com/fathurwithyou/Tucil2_13523105