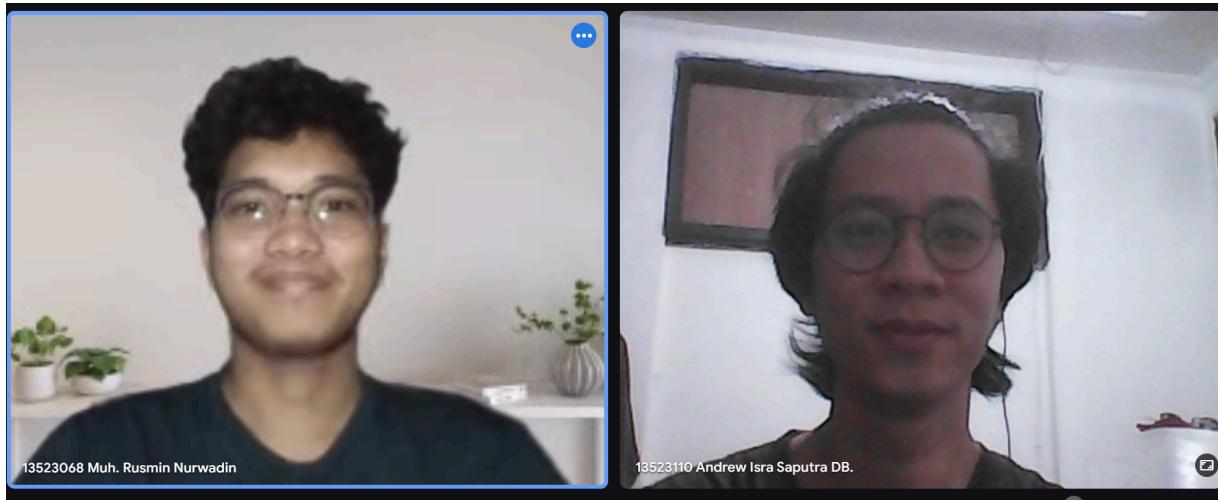


Tugas Kecil 2 IF2211 Strategi Algoritma

Kompresi Gambar Dengan Metode Quadtree



Disusun oleh :

Muh. Rusmin Nurwadin 13523068

Andrew Isra Saputra DB 13523110

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

Daftar Isi

Daftar Isi.....	2
Bab I.....	3
Deskripsi Masalah.....	3
1.1 Deskripsi Umum Tugas.....	3
1.2.1 Error Measurement Methods (Metode Pengukuran Error).....	3
1. Varians.....	3
2. Mean Absolute Deviation (MAD).....	4
3. Maximum Pixel Difference.....	4
4. Entropy.....	4
5. Structural Similarity Index (SSIM).....	5
1.2.2 Threshold (Ambang Batas).....	5
1.2.3 Minimum Block Size (Ukuran Blok Minimum).....	5
Bab II.....	7
Penyelesaian Dengan Divide and Conquer.....	7
2.1 Algoritma Divide and Conquer.....	7
2.2 Penyelesaian Masalah dengan Pendekatan Divide and Conquer.....	8
Bab III.....	12
Implementasi Algoritma.....	12
3.1 Image Utils.....	12
3.2 Error Metrics.....	13
3.3 Quadtree.....	18
Bab IV.....	22
Hasil dan Pembahasan.....	22
4.1 Spesifikasi Gambar Uji Coba.....	22
4.2 Hasil.....	23
4.3 Pembahasan.....	37
4.4 Testing Lainnya.....	38
BAB V.....	41
Saran dan Kesimpulan.....	41
5.1 Saran.....	41
5.2 Kesimpulan.....	41
Lampiran.....	42
Referensi.....	43

Bab I

Deskripsi Masalah

1.1 Deskripsi Umum Tugas

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 Parameter

1.2.1 Error Measurement Methods (Metode Pengukuran Error)

Dalam proses kompresi gambar menggunakan metode quadtree, salah satu aspek paling penting adalah menentukan kapan sebuah blok gambar dianggap cukup seragam, seragam disini memiliki arti komponen ketiga channel RGB (Red, Green, and Blue) memiliki nilai yang mirip. Hal ini bertujuan untuk menentukan apakah sebuah blok tidak perlu dibagi lebih lanjut. Untuk mengukur tingkat keseragaman tersebut, digunakan berbagai jenis rumus error (error formula). Rumus-rumus ini bertugas menghitung seberapa jauh nilai-nilai piksel dalam suatu blok berbeda dari nilai rata-rata atau dari satu sama lain.

Semakin kecil nilai error dalam suatu blok, maka blok tersebut dianggap lebih seragam (homogen), sehingga tidak perlu dipecah lagi. Sebaliknya, jika error lebih besar dari threshold (ambang batas) yang telah ditentukan, maka blok tersebut akan terus dibagi lagi menjadi empat bagian (sesuai konsep quadtree) hingga kondisi keseragaman terpenuhi atau ukuran minimum blok tercapai.

1. Varians

Dalam teori probabilitas dan statistika, varians (dari bahasa Inggris: variance) atau ragam suatu peubah acak (atau distribusi probabilitas) adalah ukuran seberapa jauh sebuah kumpulan bilangan tersebar. Varians nol mengindikasikan bahwa semua nilai sama.

Varians selalu bernilai non-negatif. Varians yang rendah mengindikasikan bahwa titik data condong sangat dekat dengan nilai rerata (nilai ekspektasi) dan antara satu sama lainnya, sementara varians yang tinggi mengindikasikan bahwa titik data sangat tersebar di sekitar rerata dan dari satu sama lainnya. Oleh karena itu, varians disini berguna untuk menentukan persebaran suatu data. Dalam konteks gambar, varians digunakan untuk mengetahui seberapa beragam warna dalam suatu blok piksel.

$$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$$

$$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$$

2. Mean Absolute Deviation (MAD)

MAD adalah ukuran alternatif dari sebaran data yang lebih robust terhadap outlier dibandingkan variansi. MAD menghitung rata-rata dari selisih absolut nilai piksel terhadap nilai rata-rata blok.

$$MAD_c = \frac{1}{N} \sum_{i=1}^N |P_{i,c} - \mu_c|$$

$$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$$

3. Maximum Pixel Difference

Metode ini menggunakan selisih antara nilai piksel terbesar dan terkecil dalam suatu blok untuk menentukan keseragaman.

$$D_c = \max(P_{i,c}) - \min(P_{i,c})$$

$$D_{RGB} = \frac{D_R + D_G + D_B}{3}$$

4. Entropy

Entropi adalah ukuran dari ketidakteraturan atau kompleksitas data. Dalam pengolahan citra, entropi digunakan untuk mengukur seberapa acak distribusi nilai piksel dalam suatu blok. Entropi tinggi menunjukkan bahwa nilai piksel tersebar luas dan acak, sedangkan entropi rendah menunjukkan keseragaman.

$$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$$

$$H_{RGB} = \frac{H_R + H_G + H_B}{3}$$

5. Structural Similarity Index (SSIM)

1.2.2 Threshold (Ambang Batas)

Threshold atau ambang batas adalah suatu nilai yang digunakan sebagai parameter pengontrol dalam proses pengambilan keputusan pada algoritma kompresi gambar menggunakan metode quadtree. Threshold digunakan untuk menentukan apakah suatu blok gambar cukup seragam (homogen) atau perlu dibagi lagi menjadi empat sub-blok yang lebih kecil.

Dalam hal ini, nilai threshold digunakan sebagai pembanding terhadap nilai error dari suatu blok. Jika nilai error suatu blok (yang diukur menggunakan formula seperti variansi, MAD, atau entropi) lebih kecil atau sama dengan threshold, maka blok tersebut dianggap cukup seragam dan tidak perlu dipecah lebih lanjut. Sebaliknya, jika nilai error lebih besar dari threshold, maka blok akan dipecah menjadi empat bagian (sesuai prinsip quadtree), dan proses ini diulangi secara rekursif.

1.2.3 Minimum Block Size (Ukuran Blok Minimum)

Minimum block size atau ukuran minimum blok adalah batas bawah dari ukuran sebuah blok gambar yang boleh dipecah dalam proses kompresi menggunakan algoritma Quadtree. Parameter ini digunakan untuk menghentikan rekursi pemecahan blok, terutama ketika ukuran blok sudah terlalu kecil dan tidak lagi efisien untuk diproses atau dibagi.

Secara umum, quadtree akan membagi blok gambar menjadi empat sub-blok jika blok tersebut dianggap tidak homogen (tidak seragam). Namun, pembagian tersebut tidak boleh dilakukan terus menerus tanpa batas. Oleh karena itu, ditetapkanlah nilai minimum block size untuk membatasi seberapa kecil suatu blok boleh menjadi.

Fungsi Minimum Block Size dalam Algoritma Quadtree :

1. Mencegah Overfitting Visual
Tanpa batasan ukuran blok, algoritma bisa memecah gambar hingga ke tingkat piksel tunggal (1×1), yang tentu saja akan membuat proses kompresi gagal mengurangi ukuran gambar secara efektif.
2. Mengontrol Kompleksitas dan Efisiensi Komputasi
Pemrosesan blok-blok sangat kecil secara rekursif dapat membebani kinerja program, meningkatkan kompleksitas waktu dan penggunaan memori. Minimum block size membatasi hal tersebut.
3. Menjadi Parameter Penyeimbang Kompresi dan Kualitas
Bersama threshold, parameter ini membantu menentukan seberapa besar detail yang akan dipertahankan dalam hasil kompresi. Semakin besar minimum block size, semakin kasar hasil gambar (lebih banyak informasi dikompresi); semakin kecil, semakin halus detail yang dipertahankan.

1.3 Spesifikasi Tugas

Pada tugas ini, dibuat program sederhana dalam bahasa C/C#/C++/Java (CLI) yang mengimplementasikan algoritma divide and conquer untuk melakukan kompresi gambar berbasis quadtree yang mengimplementasikan seluruh parameter yang telah disebutkan sebagai user input.

Alur program:

1. [INPUT] alamat absolut gambar yang akan dikompresi.
2. [INPUT] metode perhitungan error (gunakan penomoran sebagai input).
3. [INPUT] ambang batas (pastikan range nilai sesuai dengan metode yang dipilih).
4. [INPUT] ukuran blok minimum.
5. [INPUT] Target persentase kompresi (floating number, 1.0 = 100%), beri nilai 0 jika ingin menonaktifkan mode ini, jika mode ini aktif maka nilai threshold bisa menyesuaikan secara otomatis untuk memenuhi target persentase kompresi (bonus).
6. [INPUT] alamat absolut gambar hasil kompresi.
7. [INPUT] alamat absolut gif (bonus).
8. [OUTPUT] waktu eksekusi.
9. [OUTPUT] ukuran gambar sebelum.
10. [OUTPUT] ukuran gambar setelah.
11. [OUTPUT] persentase kompresi.
12. [OUTPUT] kedalaman pohon.
13. [OUTPUT] banyak simpul pada pohon.
14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.
15. [OUTPUT] GIF proses kompresi pada alamat yang sudah ditentukan (bonus).

Bab II

Penyelesaian Dengan Divide and Conquer

2.1 Algoritma Divide and Conquer

Algoritma Divide and Conquer merupakan salah satu metode pemecahan masalah dalam menyelesaikan persoalan sehari-hari, utamanya persoalan yang cukup besar. Algoritma tersebut terbagi menjadi dua bagian, yaitu **divide** dan **conquer**. Divide artinya membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama). Conquer artinya menyelesaikan (solve) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar), serta tahapan tambahan yaitu combine yang menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Berikut merupakan contoh ide dari metode Divide and Conquer

```
procedure DIVIDEandCONQUER(input P : problem, n : integer)
{ Menyelesaikan persoalan P dengan algoritma divide and conquer
Masukan: masukan persoalan P berukuran n Luaran: solusi dari
persoalan semula }
Deklarasi
    r : integer

Algoritma
    if n = 0 then {ukuran persoalan P sudah cukup kecil }
        SOLVE persoalan P yang berukuran n ini
    else
        DIVIDE menjadi r upa-persoalan, P1 , P2 , ..., Pr , yang
        masing-masing berukuran n1 , n2 , ..., nr

        for masing-masing P1 , P2 , ..., Pr , do
            DIVIDEandCONQUER(Pi , ni )
        endfor COMBINE solusi dari P1 , P2 , ..., Pr menjadi solusi
        persoalan semula

    endif
```

Keuntungan dari algoritma divide and conquer adalah dapat memecahkan masalah dengan lebih efisien, terutama jika masalahnya dapat dibagi menjadi sub-masalah yang lebih kecil yang dapat diselesaikan secara independen. Salah satu contoh penting adalah pada algoritma pengurutan, yang dapat mengurangi kompleksitas waktu dibandingkan dengan pendekatan yang lebih sederhana seperti pengurutan dengan metode bubble sort atau insertion sort.

Secara umum, algoritma divide and conquer memiliki kompleksitas waktu yang lebih baik, misalnya, pada merge sort dan quick sort yang memiliki kompleksitas waktu rata-rata $O(n \log n)$, yang lebih efisien dibandingkan algoritma pengurutan yang lebih sederhana yang memiliki kompleksitas $O(n^2)$.

Beberapa contoh algoritma yang menggunakan paradigma divide and conquer antara lain:

1. Merge Sort: Memecah array menjadi dua bagian, mengurutkannya, dan kemudian menggabungkan hasilnya.
2. Quick Sort: Memilih elemen pivot dan mempartisi array berdasarkan nilai pivot, kemudian mengurutkan bagian kiri dan kanan secara rekursif.
3. Binary Search: Membagi array yang terurut menjadi dua bagian dan mencari elemen di salah satu bagian.
4. Kompresi gambar dengan quadtree : Membagi gambar menjadi empat bagian terus menerus secara rekursif sampai ditemukan ambang batas yang sesuai.

2.2 Penyelesaian Masalah dengan Pendekatan Divide and Conquer

Algoritma quadtree untuk kompresi gambar bekerja dengan membagi gambar menjadi empat kuadran secara rekursif, kemudian merepresentasikan area seragam dengan satu nilai warna. Berikut penjabaran lebih lanjut terkait bagian divide, conquer, dan combine.

1. **Divide** : Gambar dibagi menjadi empat kuadran yang sama jika nilai error melebihi threshold dan ukuran blok masih di atas ukuran minimum.
2. **Conquer** : Setiap kuadran diselesaikan secara rekursif dengan algoritma yang sama, sehingga menghasilkan representasi quadtree dari setiap bagian.
3. **Combine** : Gambar akhir dihasilkan dengan menggabungkan informasi dari semua node pada quadtree, di mana setiap node daun akan digambar dengan warna rata-ratanya.

Secara umum, alur kerja dari program adalah berikut.

1. Inisialisasi
Gambar, nilai threshold, dan ukuran blok minimum diterima dari input pengguna. Kemudian menyiapkan struktur data quadtree untuk menyimpan hasil kompresi.
2. Analisis Gambar Awal
Menghitung error (variansi warna) dari seluruh gambar, lalu bandingkan dengan threshold yang ditentukan. Jika error di bawah threshold atau ukuran sudah minimal, representasikan seluruh gambar dengan satu warna
3. Divide
Hitung nilai error pada area tersebut menggunakan metode yang dipilih, seperti variansi, MAD (Mean Absolute Deviation), maximum difference, atau entropi, untuk mengukur seberapa beragam warna di dalamnya.

Jika error melebihi threshold dan ukuran blok lebih besar dari minimum, maka bagi gambar menjadi empat kuadran yang sama besar. kuadran 1 bagian kiri atas, kuadran 2 bagian kanan atas, kuadran 3 bagian kiri bawah, dan kuadran 4 bagian kanan bawah.

4. Conquer

Jika nilai error di bawah atau sama dengan threshold, atau ukuran blok sudah mencapai batas minimum, maka area tersebut dianggap cukup homogen dan tidak perlu dibelah lagi. Pada titik ini, algoritma akan menyimpan nilai warna rata-rata dari area tersebut sebagai representasi akhir.

5. Combine

Susun kembali gambar akhir dari pohon quadtree yang dihasilkan. Untuk setiap node daun, isi area sesuai dengan warna rata-rata.

6. Hasil Akhir

Gambar telah terkompresi dengan area seragam.

7. Output

Keluaran program akan menghasilkan gambar hasil kompresi, rasio kompresi, serta statistik program.

Berikut adalah pseudocode dari langkah-langkah divide and conquer yang dilakukan.

```
procedure buildQTree(input: imgData, imgW, imgH, chn, minBlk, thresh, errMethod, stats, output: QNode)

{ Membangun struktur Quadtree dari gambar menggunakan pendekatan
divide and conquer }

Deklarasi

-
Algoritma

return quadTree(imgData, imgW, imgH, chn, 0, 0, imgW, imgH,
minBlk, thresh, errMethod, stats, 0)
```

```
function quadTree(input: imgData, w, h, chn, x, y, blkW, blkH,
minBlk, thresh, errMethod, stats, depth) → QNode

{Strategi divide and conquer untuk mengubah blok gambar menjadi
node quadtree}
```

Deklarasi

```
node : QNode  
avgR, avgG, avgB : double  
err : double
```

Algoritma

```
{CONQUER - Proses blok menjadi simpul}  
  
calcAvgColor() {panggil fungsi dengan parameter terkait}  
  
{set warna rata-rata node ← avgR, avgG, avgB}  
{set koordinat node}  
  
stats.nodeCount ← stats.nodeCount + 1  
stats.maxDepth ← max(stats.maxDepth, depth) {jika depth lebih besar, update}  
  
if w > minBlk AND h > minBlk then  
  
    {CONQUER - Hitung error pada blok}  
  
    err ← calcError(imgData, w, h, chn, x, y, blkW, blkH,  
    errMethod)  
  
    if err > thresh then  
  
        {DIVIDE - Bagi blok menjadi 4 sub-blok}  
        halfW ← w / 2  
        halfH ← h / 2  
  
        node.1 ← quadTree(kuadran 1)  
  
        node.2 ← quadTree(kuadran 2)  
  
        node.3 ← quadTree(kuadran 3)  
  
        node.4 ← quadTree(kuadran 4)  
  
    endif  
  
endif  
  
return node
```

```
procedure genImage(input: node : QNode, output: outImg, imgW,  
chn)
```

```
{Menghasilkan kembali gambar dari hasil quadtree}
```

Deklarasi

```
-
```

Algoritma

```
if node tidak memiliki anak (leaf node) then

{COMBINE - Warnai blok dengan warna node}

    for i ← y to y + h - 1 do
        for j ← x to x + w - 1 do
            setPixel() {memanggil setPixel, dengan parameter
            tertentu}
        endfor
    endfor

else

    {REKURSIF ke semua anak}
    genImage(node.1)
    genImage(node.2)
    genImage(node.3)
    genImage(node.4)

endif
```

Dalam kasus terbaik, saat gambar homogen, seluruh proses hanya dilakukan satu kali dan menghasilkan kompleksitas $O(n)$, dengan n adalah jumlah piksel. Namun, pada kasus terburuk ketika gambar sangat kompleks dan setiap piksel perlu diproses sebagai simpul tersendiri, kompleksitas meningkat hingga $O(n^2)$ karena setiap pembagian blok melibatkan perhitungan error dan rata-rata warna yang bersifat linier terhadap ukuran blok.

Rata-rata kompleksitas waktu dari program ini berada di antara $O(n \log n)$, terutama untuk gambar alami dengan detail yang bervariasi di beberapa bagian saja. Proses pembuatan pohon quadtree yang adaptif terhadap nilai ambang batas menjadikan algoritma ini efisien untuk kompresi gambar berbasis konten visual, sementara penggunaan struktur rekursif memungkinkan penyimpanan informasi gambar secara hierarkis dan hemat memori.

Bab III

Implementasi Algoritma

Implementasi algoritma dilakukan dengan menggunakan bahasa C++. Dengan file sebagai berikut.

3.1 Image Utils

File `image_utils.cpp` berisi implementasi fungsi-fungsi utilitas dasar untuk manipulasi gambar yang digunakan dalam proses kompresi berbasis quadtree. File ini terdiri atas beberapa fungsi/prosedur berikut.

1. `getPixel`

Berfungsi untuk mengambil nilai RGB dari piksel tertentu dalam gambar.

```
1 void getPixel(unsigned char *data, int imgW, int x, int y, int chn, int &r, int &g, int &b) {
2     // Validasi batas
3     if (x < 0 || y < 0 || x >= imgW) {
4         r = g = b = 0;
5         return;
6     }
7     int idx = (y * imgW + x) * chn;
8     r = data[idx];
9     g = data[idx + 1];
10    b = data[idx + 2];
11 }
12 }
```

2. `setPixel`

berfungsi untuk menyetel nilai RGB pada pixel tertentu.

```
1 void setPixel(unsigned char *data, int imgW, int x, int y, int chn, int r, int g, int b) {
2     int idx = (y * imgW + x) * chn;
3     data[idx] = r;
4     data[idx + 1] = g;
5     data[idx + 2] = b;
6 }
```

3. calcAvgColor

Berfungsi untuk menghitung rata-rata warna (RGB) dalam sebuah blok gambar

```
1 void calcAvgColor(unsigned char *imgData, int imgW, int imgH, int chn, int x, int y,
2     int w, int h, int &r, int &g, int &b) {
3     long sumR = 0, sumG = 0, sumB = 0;
4     int count = 0;
5
6     for (int j = y; j < y + h && j < imgH; j++) {
7         for (int i = x; i < x + w && i < imgW; i++) {
8             int pixR, pixG, pixB;
9             getPixel(imgData, imgW, i, j, chn, pixR, pixG, pixB);
10            sumR += pixR;
11            sumG += pixG;
12            sumB += pixB;
13            count++;
14        }
15
16        if (count > 0) {
17            r = static_cast<int>(sumR / count);
18            g = static_cast<int>(sumG / count);
19            b = static_cast<int>(sumB / count);
20        } else {
21            r = g = b = 255;
22        }
23 }
```

4. calcImgSize

Berfungsi untuk menghitung total ukuran gambar dalam byte berdasarkan dimensi dan jumlah channel.

```
1 size_t calcImgSize(int w, int h, int chn) {
2     return static_cast<size_t>(w) * h * chn;
3 }
```

3.2 Error Metrics

File ini (error_metrics.cpp) berisi implementasi berbagai metode untuk mengukur tingkat ketidakhomogenan atau *error* dalam sebuah blok gambar. Ini berguna untuk menentukan apakah suatu

blok layak dipecah lebih lanjut dalam struktur quadtree saat kompresi. File ini terbagi atas fungsi/prosedur berikut.

1. calcVar (Variansi)

Mengukur variasi warna dalam blok. Semakin besar nilai variansi, semakin tidak homogen warnanya.

```
1 // Menghitung variansi warna dalam blok gambar
2 double calcVar(unsigned char *imgData, int imgW, int imgH, int chn, int x, int y, int w, int h)
{
3     double mean[3] = {0.0, 0.0, 0.0};
4     double var[3] = {0.0, 0.0, 0.0};
5     int count = 0;
6
7     // Menghitung rata-rata tiap channel dalam blok gambar
8     for (int i = y; i < y + h; i++) {
9         for (int j = x; j < x + w; j++) {
10            int idx = (i * imgW + j) * chn;
11            for (int c = 0; c < chn; c++) {
12                mean[c] += imgData[idx + c];
13            }
14            count++;
15        }
16    }
17    for (int c = 0; c < chn; c++) {
18        mean[c] /= count;
19    }
20
21    // Menghitung variansi tiap channel dalam blok gambar
22    for (int i = y; i < y + h; i++) {
23        for (int j = x; j < x + w; j++) {
24            int idx = (i * imgW + j) * chn;
25            for (int c = 0; c < chn; c++) {
26                var[c] += pow(imgData[idx + c] - mean[c], 2);
27            }
28        }
29    }
30    for (int c = 0; c < chn; c++) {
31        var[c] /= count;
32    }
33
34    double sumOfSquares = 0.0;
35    for (int c = 0; c < chn; c++) {
36        sumOfSquares += pow(var[c], 2); // Menghitung jumlah kuadrat dari variansi tiap channel
37    }
38
39    double meanOfSquares = sumOfSquares / chn; // Menghitung rata-rata kuadrat dari variansi
40    return meanOfSquares;
41 }
```

2. calcMad (Mean Absolute Deviation)

Mengukur seberapa jauh rata-rata nilai piksel menyimpang dari nilai rata-rata keseluruhan. Mirip variansi, tapi pakai nilai absolut (lebih ringan dihitung dan lebih tahan terhadap outlier).

```
 1 // Menghitung Mean Absolute Deviation dalam blok gambar
 2 double calcMAD(unsigned char *imgData, int imgW, int imgH, int chn, int x, int y, int w, int h){
 3     double mean[3] = {0.0, 0.0, 0.0};
 4     double mad[3] = {0.0, 0.0, 0.0};
 5     int count = 0;
 6
 7     // Menghitung rata-rata tiap channel dalam blok gambar
 8     for (int i = y; i < y + h; i++) {
 9         for (int j = x; j < x + w; j++) {
10             int idx = (i * imgW + j) * chn;
11             for (int c = 0; c < chn; c++) {
12                 mean[c] += imgData[idx + c];
13             }
14             count++;
15         }
16     }
17     for (int c = 0; c < chn; c++) {
18         mean[c] /= count;
19     }
20
21     // Menghitung Mean Absolute Deviation tiap channel dalam blok gambar
22     for (int i = y; i < y + h; i++) {
23         for (int j = x; j < x + w; j++) {
24             int idx = (i * imgW + j) * chn;
25             for (int c = 0; c < chn; c++) {
26                 mad[c] += fabs(imgData[idx + c] - mean[c]);
27             }
28         }
29     }
30     for (int c = 0; c < chn; c++) {
31         mad[c] /= count;
32     }
33
34     double sumOfAbsDiffs = 0.0;
35     for(int c = 0; c < chn; c++) {
36         sumOfAbsDiffs += mad[c]; // Menghitung jumlah dari MAD tiap channel
37     }
38
39     double meanOfAbsDiffs = sumOfAbsDiffs / chn; // Menghitung rata-rata dari MAD
40     return meanOfAbsDiffs;
41 }
```

3. calcMaxDiff

Mengukur perbedaan maksimum antar piksel pada tiap channel (R, G, B). Metode paling simpel dan cepat.

```
1 // Menghitung perbedaan maksimum piksel dalam blok gambar
2 double calcMaxDiff(unsigned char *imgData, int imgW, int imgH, int chn, int x, int y, int w, int h){
3     int minVal[3] = {255, 255, 255};
4     int maxVal[3] = {0, 0, 0};
5
6     // Menghitung nilai minimum dan maksimum tiap channel dalam blok gambar
7     for (int i = y; i < y + h; i++) {
8         for (int j = x; j < x + w; j++) {
9             int idx = (i * imgW + j) * chn;
10            for (int c = 0; c < chn; c++) {
11                int val = imgData[idx + c];
12                minVal[c] = min(minVal[c], val);
13                maxVal[c] = max(maxVal[c], val);
14            }
15        }
16    }
17
18    double sumOfMaxDiffs = 0.0;
19    for (int c = 0; c < chn; c++) {
20        double maxDiff = maxVal[c] - minVal[c]; // Menghitung perbedaan maksimum tiap channel
21        sumOfMaxDiffs += maxDiff; // Menghitung jumlah dari perbedaan maksimum tiap channel
22    }
23
24    double meanOfMaxDiffs = sumOfMaxDiffs / chn; // Menghitung rata-rata dari perbedaan maksimum
25    return meanOfMaxDiffs;
26 }
```

4. calcEntropy

Mengukur kompleksitas informasi dalam blok berdasarkan distribusi frekuensi warna. Cocok untuk deteksi tekstur/detail.

```
1 // Menghitung entropi warna dalam blok gambar
2 double calcEntropy(unsigned char *imgData, int imgW, int imgH, int chn, int x, int y, int w, int h){
3     int freqR[256] = {0};
4     int freqG[256] = {0};
5     int freqB[256] = {0};
6     int total = w * h;
7
8     for (int i = y; i < y + h; ++i) {
9         for (int j = x; j < x + w; ++j) {
10             int idx = (i * imgW + j) * chn;
11             freqR[imgData[idx]]++;
12             freqG[imgData[idx + 1]]++;
13             freqB[imgData[idx + 2]]++;
14         }
15     }
16
17     auto calcOneEntropy = [](int freq[256], int total) -> double {
18         double entropy = 0.0;
19         for (int k = 0; k < 256; ++k) {
20             if (freq[k] > 0) {
21                 double p = (double)freq[k] / total;
22                 entropy -= p * log2(p);
23             }
24         }
25         return entropy;
26     };
27
28     double entropyR = calcOneEntropy(freqR, total);
29     double entropyG = calcOneEntropy(freqG, total);
30     double entropyB = calcOneEntropy(freqB, total);
31
32     return (entropyR + entropyG + entropyB) / 3.0;
33 }
```

5. calcError

Fungsi pemilih metode error di atas berdasarkan parameter input method (1–4)

```
1 // Memilih dan memanggil metode pengukuran error yang sesuai
2 double calcError(unsigned char* imgData, int imgW, int imgH, int chn, int x, int y, int w, int h,
3                  int method){
4     switch (method) {
5         case 1:
6             return calcVar(imgData, imgW, imgH, chn, x, y, w, h);
7         case 2:
8             return calcMAD(imgData, imgW, imgH, chn, x, y, w, h);
9         case 3:
10            return calcMaxDiff(imgData, imgW, imgH, chn, x, y, w, h);
11        case 4:
12            return calcEntropy(imgData, imgW, imgH, chn, x, y, w, h);
13        default:
14            return -1.0; // Error code
15    }
16 }
```

3.3 Quadtree

File ini berisi implementasi struktur data Quadtree untuk membagi gambar ke dalam blok-blok berdasarkan tingkat homogenitas warna. Tujuannya adalah untuk merepresentasikan gambar secara efisien dengan membagi hanya bagian yang memiliki variasi warna signifikan. Berikut penjabaran fungsi/prosedur terkait.

1. quadTree

Fungsi rekursif utama untuk membangun simpul-simpul quadtree. Jika blok saat ini lebih besar dari ukuran minimum dan nilai error-nya melebihi ambang batas, maka blok akan dipecah menjadi empat bagian. Setiap simpul menyimpan koordinat, ukuran, warna rata-rata, dan pointer ke anak-anaknya.

```

1 QNode *quadTree(unsigned char *imgData, int imgW, int imgH, int chn, int x, int y, int w, int h, int minBlk, double thresh, int
errMethod, QStats &stats, int depth) {
2     QNode *node = new QNode();
3     node->x = x;
4     node->y = y;
5     node->w = w;
6     node->h = h;
7
8     calcAvgColor(imgData, imgW, imgH, chn, x, y, w, h, node->r, node->g, node->b);
9
10    stats.nodeCount++;
11    stats.maxDepth = max(stats.maxDepth, depth);
12
13    if (w > minBlk && h > minBlk) {
14        double error = calcError(imgData, imgW, imgH, chn, x, y, w, h, errMethod);
15
16        if (error > thresh) {
17            int halfW = w / 2;
18            int halfH = h / 2;
19
20            int w1 = halfW;
21            int w2 = w - halfW;
22            int h1 = halfH;
23            int h2 = h - halfH;
24
25            node->child[0] = quadTree(imgData, imgW, imgH, chn, x, y, w1, h1, minBlk, thresh, errMethod, stats, depth + 1);
26            node->child[1] = quadTree(imgData, imgW, imgH, chn, x + w1, y, w2, h1, minBlk, thresh, errMethod, stats, depth + 1);
27            node->child[2] = quadTree(imgData, imgW, imgH, chn, x, y + h1, w1, h2, minBlk, thresh, errMethod, stats, depth + 1);
28            node->child[3] = quadTree(imgData, imgW, imgH, chn, x + w1, y + h1, w2, h2, minBlk, thresh, errMethod, stats, depth + 1);
29        }
30    }
31
32    return node;
33 }
```

2. buildQTree

Fungsi untuk membangun quadtree dari seluruh gambar. Fungsi ini mengatur ulang statistik (jumlah node dan kedalaman maksimum), lalu memanggil quadTree mulai dari koordinat (0, 0) dengan lebar dan tinggi penuh gambar.

```

1 QNode *buildQTree(unsigned char *imgData, int imgW, int imgH, int chn, int minBlk, double thresh, int errMethod, QStats& stats) {
2     stats.nodeCount = 0;
3     stats.maxDepth = 0;
4     return quadTree(imgData, imgW, imgH, chn, 0, 0, imgW, imgH, minBlk, thresh, errMethod, stats, 0);
5 }
```

3. genImage

Membentuk ulang gambar hasil kompresi dari struktur quadtree. Jika sebuah simpul adalah daun (tidak memiliki anak), maka area yang diwakilinya akan diwarnai dengan warna rata-ratanya. Jika simpul memiliki anak, fungsi akan memanggil dirinya sendiri secara rekursif untuk setiap anak.

```
1 void genImage(QNode *root, unsigned char *outImg, int imgW, int chn) {
2     if (!root){
3         return;
4     }
5
6     if (!root->child[0] && !root->child[1] && !root->child[2] && !root->child[3]) {
7         //COMBINE
8         for (int y = root->y; y < root->y + root->h; y++) {
9             for (int x = root->x; x < root->x + root->w; x++) {
10                 if (x >= 0 && x < imgW && y >= 0) {
11                     setPixel(outImg, imgW, x, y, chn, root->r, root->g, root->b);
12                 }
13             }
14         }
15     } else {
16         //COMBINE
17         for (int i = 0; i < 4; i++) {
18             if (root->child[i]) {
19                 genImage(root->child[i], outImg, imgW, chn);
20             }
21         }
22     }
23 }
24 }
```

4. **cleanQTree**

Membersihkan dan menghapus semua node dari memori secara rekursif. Fungsi ini akan memanggil cleanQTree untuk semua anak yang ada, lalu menghapus simpul induknya sendiri.



```
1 void cleanQTree(QNode *root) {  
2     if (!root){  
3         return;  
4     }  
5     for (int i = 0; i < 4; i++) {  
6         if (root->child[i]) {  
7             cleanQTree(root->child[i]);  
8         }  
9     }  
10  
11     delete root;  
12 }
```

Bab IV

Hasil dan Pembahasan

4.1 Spesifikasi Gambar Uji Coba

Gambar yang digunakan merupakan gambar dengan keragaman warna yang tinggi serta ukuran file yang tergolong besar. Gambar testing yang digunakan menggunakan tiga channel (RGB).



testing.jpg



testing2.jpg



testing3.jpg

4.2 Hasil

Berikut merupakan hasil untuk masing-masing metode dengan threshold yang sesuai.

1. Metode Varians

Threshold 1000, ukuran blok minimum 4.



```
Masukkan path gambar input (ex: input.jpg): testing.jpg
Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 1
Nilai ambang batas (threshold): 1000
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_varians1.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_varians1.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 48721 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 13861804 bytes
Persentase kompresi: 80.7475%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 315041
PS E:\Kodingan\Semester 4\Stima\Tucil2\Tucil2_13523068_13523110>
```

Threshold 10.000, ukuran blok minimum 4.



```
Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 1
Nilai ambang batas (threshold): 10000
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_varians2.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_varians2.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 59787 ms
Ukuran gambar asli: 7200000 bytes
Ukuran gambar terkompresi: 7849996 bytes
Persentase kompresi: 89.0972%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 178409
```

Threshold 400, ukuran blok minimum 4.



```
Masukkan path gambar input (ex: input.jpg): testing.jpg
Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 1
Nilai ambang batas (threshold): 400
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_varians3.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_varians3.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 53391 ms
Ukuran gambar asli: 7200000 bytes
Ukuran gambar terkompresi: 18039692 bytes
Persentase kompresi: 74.9449%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 409993
```

Threshold 1000, ukuran blok minimum 16



```
Masukkan path gambar input (ex: input.jpg): testing.jpg
Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 1
Nilai ambang batas (threshold): 1000
Ukuran blok minimum: 16
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_varians5.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_varians5.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 43811 ms
Ukuran gambar asli: 7200000 bytes
Ukuran gambar terkompresi: 1606396 bytes
Persentase kompresi: 97.7689%
Kedalaman pohon: 8
Jumlah simpul pada pohon: 36509
```

Threshold 1000, ukuran blok minimum 64

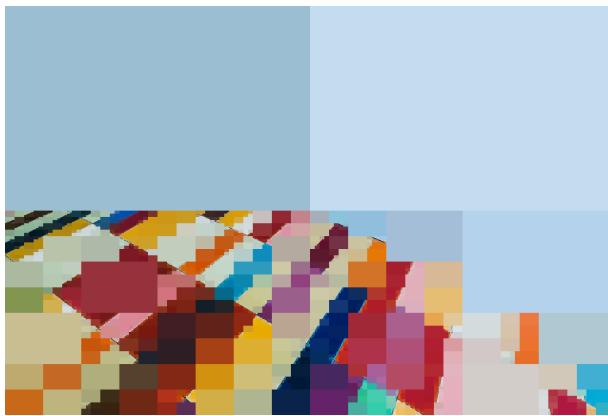


```
Masukkan path gambar input (ex: input.jpg): testing.jpg
Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 1
Nilai ambang batas (threshold): 1000
Ukuran blok minimum: 64
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_varians4.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_varians4.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 44959 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 169004 bytes
Persentase kompresi: 99.7653%
Kedalaman pohon: 6
Jumlah simpul pada pohon: 3841
```

2. Metode Mean Absolute Deviation (MAD)

Threshold 40, ukuran blok minimum 4.



```

Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 2
Nilai ambang batas (threshold): 40
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_MAD1.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_MAD1.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 3220 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 268268 bytes
Persentase kompresi: 99.6274%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 609

```

Threshold 20, ukuran blok minimum 4.



```

Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 2
Nilai ambang batas (threshold): 20
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_MAD2.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_MAD2.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 4818 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 2270796 bytes
Persentase kompresi: 96.8461%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 51609

```

Threshold 5, ukuran blok minimum 4.



```
Masukkan path gambar input (ex: input.jpg): testing.jpg
Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 2
Nilai ambang batas (threshold): 5
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_MAD3.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_MAD3.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 6504 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 11068860 bytes
Persentase kompresi: 84.6266%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 251565
```

Threshold 10, ukuran blok minimum 16.



```

Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 2
Nilai ambang batas (threshold): 5
Ukuran blok minimum: 16
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_MAD4.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_MAD4.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 6189 ms
Ukuran gambar asli: 7200000 bytes
Ukuran gambar terkompresi: 1282732 bytes
Persentase kompresi: 98.2184%
Kedalaman pohon: 8
Jumlah simpul pada pohon: 29153

```

Threshold 5, ukuran blok minimum 64.



```

Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 2
Nilai ambang batas (threshold): 5
Ukuran blok minimum: 64
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_MAD5.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_MAD5.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 5448 ms
Ukuran gambar asli: 7200000 bytes
Ukuran gambar terkompresi: 151756 bytes
Persentase kompresi: 99.7892%
Kedalaman pohon: 6
Jumlah simpul pada pohon: 3449

```

3. Metode Max Pixel Difference

Threshold 30, ukuran blok minimum 4.



```
Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 3
Nilai ambang batas (threshold): 30
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_MaxDiff1.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_MaxDiff1.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 6180 ms
Ukuran gambar asli: 7200000 bytes
Ukuran gambar terkompresi: 12914748 bytes
Persentase kompresi: 82.0628%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 29351
```

Threshold 10, ukuran blok minimum 4.



```

Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 3
Nilai ambang batas (threshold): 10
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_MaxDiff2.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_MaxDiff2.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 8124 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 40088796 bytes
Persentase kompresi: 44.3211%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 911109

```

Threshold 60, ukuran blok minimum 4.



```

Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 3
Nilai ambang batas (threshold): 60
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_MaxDiff3.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_MaxDiff3.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 5515 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 6267404 bytes
Persentase kompresi: 91.2953%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 142441

```

Threshold 30, ukuran blok minimum 16.



```
Masukkan path gambar input (ex: input.jpg): testing.jpg  
Metode perhitungan error.  
1. Variance  
2. MAD  
3. Max Diff  
4. Entropi  
Pilih metode perhitungan error (1/2/3/4): 3  
Nilai ambang batas (threshold): 30  
Ukuran blok minimum: 16  
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0  
Masukkan path gambar output (ex: hasil.png): hasil_MaxDiff4.jpg  
Gambar berhasil dimuat: 6000x4000 dengan 3 channel  
Gambar berhasil disimpan di: test/hasil_MaxDiff4.jpg  
  
===== HASIL KOMPRESI =====  
Waktu eksekusi: 6097 ms  
Ukuran gambar asli: 7200000 bytes  
Ukuran gambar terkompresi: 1749660 bytes  
Persentase kompresi: 97.5699%  
Kedalaman pohon: 8  
Jumlah simpul pada pohon: 39765
```

Threshold 30, ukuran blok minimum 64.



```

Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 3
Nilai ambang batas (threshold): 30
Ukuran blok minimum: 64
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_MaxDiff4.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_MaxDiff4.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 5198 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 184668 bytes
Persentase kompresi: 99.7435%
Kedalaman pohon: 6
Jumlah simpul pada pohon: 4197

```

4. Metode Entropy

Threshold 0.4, ukuran blok minimum 9.



```

Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 4
Nilai ambang batas (threshold): 0.4
Ukuran blok minimum: 9
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_entropi1.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_entropi1.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 5790 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 15349356 bytes
Persentase kompresi: 78.6814%
Kedalaman pohon: 9
Jumlah simpul pada pohon: 348849

```

Threshold 0.1, ukuran blok minimum 9.



```
Masukkan path gambar input (ex: input.jpg): testing.jpg
Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 4
Nilai ambang batas (threshold): 0.1
Ukuran blok minimum: 9
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_entropi2.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_entropi2.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 6027 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 15367660 bytes
Persentase kompresi: 78.656%
Kedalaman pohon: 9
Jumlah simpul pada pohon: 349265
```

Threshold 1, ukuran blok minimum 9.



```

PS E:\Kodingan\Semester 4\Stima\Tucil2\Tucil2_13523068_13523110> .
Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 4
Nilai ambang batas (threshold): 1
Ukuran blok minimum: 9
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_entropi3.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_entropi3.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 6682 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 15236364 bytes
Persentase kompresi: 78.8384%
Kedalaman pohon: 9
Jumlah simpul pada pohon: 346281

```

Threshold 0.4, ukuran blok minimum 16.



```

Masukkan path gambar input (ex: input.jpg): testing.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 4
Nilai ambang batas (threshold): 0.4
Ukuran blok minimum: 16
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_entropi4.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_entropi4.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 5079 ms
Ukuran gambar asli: 72000000 bytes
Ukuran gambar terkompresi: 3843004 bytes
Persentase kompresi: 94.6625%
Kedalaman pohon: 8
Jumlah simpul pada pohon: 87341

```

Threshold 0.4, ukuran blok minimum 16.



```
Masukkan path gambar input (ex: input.jpg): testing.jpg
Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 4
Nilai ambang batas (threshold): 0.4
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_entropi5.jpg
Gambar berhasil dimuat: 6000x4000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_entropi5.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 8873 ms
Ukuran gambar asli: 7200000 bytes
Ukuran gambar terkompresi: 61128540 bytes
Persentase kompresi: 15.0992%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 1389285
```

5. Menggunakan gambar dengan warna kompleks dan sederhana
Gambar kompleks



```

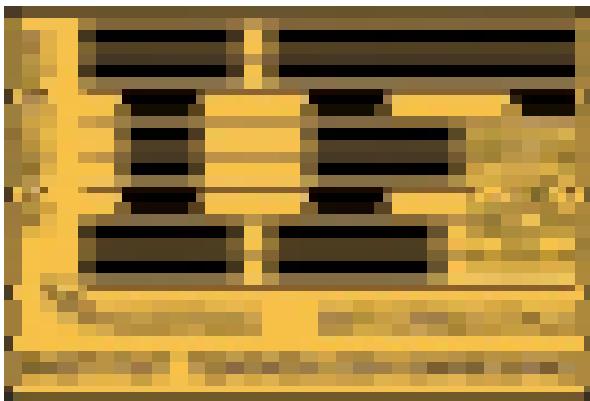
Masukkan path gambar input (ex: input.jpg): testing2.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 1
Nilai ambang batas (threshold): 1000
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_testing2.jpg
Gambar berhasil dimuat: 3008x2000 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_testing2.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 12054 ms
Ukuran gambar asli: 18048000 bytes
Ukuran gambar terkompresi: 9498412 bytes
Persentase kompresi: 47.3714%
Kedalaman pohon: 9
Jumlah simpul pada pohon: 215873

```

Gambar sederhana



```

Masukkan path gambar input (ex: input.jpg): testing3.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 1
Nilai ambang batas (threshold): 1000
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_testing3.jpg
Gambar berhasil dimuat: 200x133 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_testing3.jpg

===== HASIL KOMPRESI =====
Waktu eksekusi: 40 ms
Ukuran gambar asli: 79800 bytes
Ukuran gambar terkompresi: 86636 bytes
Persentase kompresi: -8.56642%
Kedalaman pohon: 6
Jumlah simpul pada pohon: 1969

```

4.3 Pembahasan

Terdapat empat metode yang digunakan untuk mengompresi tiga jenis gambar. Dimulai dari gambar testing.jpg yang memiliki karakteristik full warna dengan beberapa bagian memiliki variasi warna yang statis dan sebagian lain dinamis. Masing-masing metode dilakukan dengan lima percobaan yang memiliki threshold dan minimum blok yang bervariasi

Pada percobaan keempat metode, diperoleh hasil bahwa semakin tinggi sebuah threshold yang dimasukkan, semakin rendah resolusi gambarnya. Hal ini disebabkan oleh algoritma *divide* yang akan semakin panjang (rekursif terus menerus) jika sebuah threshold yang diberikan rendah. Threshold yang rendah mengakibatkan “standar” sebuah blok dikatakan seragam semakin tinggi (sukar dikatakan seragam). Sebaliknya, threshold yang tinggi mengakibatkan *looping divide* semakin pendek karena “standar” yang rendah.

Adapun faktor kedua yang memengaruhi adalah nilai minimum blok yang diberikan. Semakin kecil sebuah nilai minimum blok (minimal 1 blok pixel), semakin panjang kemungkinan *looping divide* algoritmanya. Semakin tinggi sebuah nilai minimum blok, semakin pendek looping divide algoritmanya – akibat dari pembatasan maksimal node yang dapat diselami oleh quadtree.

Namun nilai minimum blok menjadi faktor kedua yang memengaruhi hasil. Faktor utama yang menentukan resolusi adalah threshold. Kendati demikian, pada percobaan yang telah dilakukan, diperoleh fakta bahwa kenaikan sedikit saja nilai minimum blok, hasil yang diperoleh juga berubah signifikan (lihat pada hasil percobaan minimum blok 4, 16, dan 64)

Waktu eksekusi dan kedalaman simpul yang ditelusuri juga menjadi fakta yang diperoleh sesuai dengan teori yang sejalan. Terlihat pada hasil bagian 5 "Menggunakan gambar dengan warna kompleks dan sederhana", waktu eksekusi gambar yang sederhana 40 ms dan gambar yang memiliki warna yang sangat kompleks 12054 ms. Dengan masing-masing jumlah simpul 215873 dan 1969. Hal ini membuktikan bahwa gambar yang lebih kompleks memiliki pemecahan (*divide*) blok lebih banyak dibanding gambar sederhana karena perbedaan warna yang signifikan sehingga sangat sulit untuk mencapai nilai threshold yang ditentukan. Salah satu faktor yang membatasi pemecahan (*divide*) ini adalah nilai minimum blok tadi yaitu 4.

Dari hasil tersebut diperoleh juga fakta untuk nilai threshold dan nilai minimum blok yang baik sebagai berikut :

1. Varians : threshold 1000, minimum blok 4
2. MAD : threshold 5, minimum blok 4
3. Max Difference : threshold 30, minimum blok 4
4. Entropi : threshold 0.4, minimum blok 9

4.4 Testing Lainnya

1. Berhasil di run pada wsl (linux)

```
rusmin@Rusmin:/mnt/d/SEMESTER 4/STIMA/TUCIL 2/Tucil2$ make run
./bin/main
Masukkan path gambar input (ex: input.jpg): flowers.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 2
Nilai ambang batas (threshold): 10
Ukuran blok minimum: 4
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil_linux.png
Gambar berhasil dimuat: 3322x4878 dengan 3 channel
Gambar berhasil disimpan di: test/hasil_linux.png

===== HASIL KOMPRESI =====
Waktu eksekusi: 2984 ms
Ukuran gambar asli: 48614148 bytes
Ukuran gambar terkompresi: 5877312 bytes
Persentase kompresi: 87.9103%
Kedalaman pohon: 10
Jumlah simpul pada pohon: 91833
```

2. Input gambar yang tidak ada pada folder test

```
PS D:\SEMESTER 4\STIMA\TUCIL 2\Tucil2> bin\main
Masukkan path gambar input (ex: input.jpg): tidakada.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 2
Nilai ambang batas (threshold): 10
Ukuran blok minimum: 2
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil.png
Gagal memuat gambar! (perhatikan path atau format gambar)
PS D:\SEMESTER 4\STIMA\TUCIL 2\Tucil2> |
```

3. Input bukan gambar

```
PS D:\SEMESTER 4\STIMA\TUCIL 2\Tucil2> bin\main
Masukkan path gambar input (ex: input.jpg): salah.txt

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 2
Nilai ambang batas (threshold): 4
Ukuran blok minimum: 2
Target persentase kompresi (0.0 - 1.0, 0 untuk menonaktifkan): 0
Masukkan path gambar output (ex: hasil.png): hasil.png
Gagal memuat gambar! (perhatikan path atau format gambar)
PS D:\SEMESTER 4\STIMA\TUCIL 2\Tucil2> |
```

4. Input tidak sesuai perintah atau format yang benar

Diluar pilihan dan format bukan integer

```
PS D:\SEMESTER 4\STIMA\TUCIL 2\Tucil2> bin\main
Masukkan path gambar input (ex: input.jpg): flowers.jpg

Metode perhitungan error.
1. Varian
2. MAD
3. Max Diff
4. Entropi
Pilih metode perhitungan error (1/2/3/4): 5
Input tidak valid. Silakan masukkan angka 1 hingga 4.

Pilih metode perhitungan error (1/2/3/4): a
Input tidak valid. Masukkan angka bulat antara 1 hingga 4.
```

BAB V

Saran dan Kesimpulan

5.1 Saran

Dalam mengerjakan tugas ini, kami menyadari beberapa hal yang dapat ditingkatkan ke depannya:

1. **Manajemen waktu** perlu ditingkatkan agar semua spesifikasi, termasuk bonus, bisa diselesaikan tepat waktu tanpa terburu-buru di akhir.
2. **Penerapan struktur modular** dalam penulisan kode terbukti sangat membantu efisiensi dan kemudahan debugging.
3. **Perencanaan awal** seperti membagi tugas dan menentukan flow kerja sejak awal akan menghindari kebingungan di tengah pengerjaan.
4. **Perbanyak testing** agar program yang dihasilkan terjamin dari bug atau masalah di kemudian hari.

5.2 Kesimpulan

Program ini menggunakan pendekatan *divide and conquer* melalui struktur data *quadtree* untuk melakukan kompresi gambar. Konsep dasarnya adalah memecah (*divide*) gambar menjadi blok-blok kecil berdasarkan homogenitas warnanya, lalu menguji tiap blok apakah masih bisa dibagi lebih lanjut sesuai nilai ambang batas yang ditentukan. Jika blok dianggap cukup homogen, maka proses berhenti (*conquer*) di area tersebut. Proses ini terus dilakukan secara rekursif, sehingga menghasilkan pohon kuadran yang merepresentasikan pembagian spasial gambar berdasarkan kompleksitas lokal.

Setiap simpul pada pohon mewakili sebuah area tertentu dari gambar. Jika suatu area memiliki variasi warna yang tinggi, maka area tersebut akan dipecah lagi menjadi empat bagian yang lebih kecil, dan begitu seterusnya hingga mencapai ukuran blok minimum atau memenuhi syarat keseragaman. Metode perhitungan error seperti variansi, MAD, perbedaan maksimum, dan entropi digunakan sebagai dasar pengambilan keputusan dalam proses pembagian ini. Inilah inti dari strategi *divide and conquer*: memecah masalah besar (seluruh gambar) menjadi sub-masalah kecil (blok-blok gambar), menyelesaiakannya secara lokal, lalu menggabungkan hasilnya menjadi solusi global dalam bentuk gambar terkompresi.

Dengan menerapkan *divide and conquer*, program ini tidak hanya efisien dalam penggunaan memori dan waktu, tetapi juga adaptif terhadap kompleksitas gambar. Area dengan detail tinggi dikompresi secara halus dengan banyak pembagian, sementara area yang lebih polos disimpan sebagai blok besar tanpa banyak pemrosesan. Hal ini menciptakan keseimbangan antara ukuran file dan kualitas visual, menjadikan algoritma ini sangat cocok untuk aplikasi pengolahan citra dengan kebutuhan kompresi cerdas.

Lampiran

Tautan Github : https://github.com/Rusmn/Tucil2_13523068_13523110

Tabel :

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

Referensi

How To Compute RGB Image Standard Deviation From Channels Statistics

<https://www.odelama.com/data-analysis/How-to-Compute-RGB-Image-Standard-Deviation-from-Channels-Statistics/>

<https://www.scitepress.org/papers/2013/42105/42105.pdf>

Image quality assessment: from error visibility to structural similarity

<https://ieeexplore.ieee.org/document/1284395>

https://www.itu.int/dms_pubrec/itu-r/rec/bt/r-rec-bt.601-7-201103-i!!pdf-e.pdf

Algoritma Divide and Conquer

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

C++ Reference

<https://en.cppreference.com/w/>