

**IF2240 Strategi Algoritma**  
**Tugas Kelompok Entity Relation**

**Jawaban Tugas 4/11/2025**

Disusun untuk memenuhi tugas mata kuliah IF2240 Strategi Algoritma pada  
Semester 2 Tahun Akademik 2024/2025



Disusun oleh:

Rhio Bimo Prakoso S (13523123)  
Arlow Emmanuel H (13523161)

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUSI TEKNOLOGI BANDUNG**

**2024**

## DAFTAR ISI

|                                                                          |           |
|--------------------------------------------------------------------------|-----------|
| <b>DAFTAR ISI.....</b>                                                   | <b>2</b>  |
| <b>DAFTAR GAMBAR.....</b>                                                | <b>3</b>  |
| <b>BAB I</b>                                                             |           |
| <b>DESKRIPSI MASALAH.....</b>                                            | <b>4</b>  |
| <b>BAB II</b>                                                            |           |
| <b>TEORI SINGKAT.....</b>                                                | <b>6</b>  |
| 2.1. Divide and Conquer.....                                             | 6         |
| 2.2. Quadtree.....                                                       | 7         |
| <b>BAB III</b>                                                           |           |
| <b>METODE PENYELESAIAN.....</b>                                          | <b>8</b>  |
| 3.1. Divide (Pembagian).....                                             | 8         |
| 3.2. Conquer/Solve (Penyelesaian).....                                   | 8         |
| 3.3. Combine (Penggabungan).....                                         | 8         |
| <b>BAB IV</b>                                                            |           |
| <b>IMPLEMENTASI.....</b>                                                 | <b>10</b> |
| 4.1. Implementasi Program.....                                           | 10        |
| 4.1.1. Error Calculation.....                                            | 10        |
| 4.1.2. Image Compressor.....                                             | 11        |
| 4.1.3. Tree.....                                                         | 13        |
| 4.1.4. Util.....                                                         | 15        |
| 4.2. Kode Sumber.....                                                    | 16        |
| 4.2.1. Error Calculation.....                                            | 16        |
| 4.2.2. Image Compressor.....                                             | 21        |
| 4.2.3. Tree.....                                                         | 29        |
| 4.2.4. Util.....                                                         | 32        |
| <b>BAB V</b>                                                             |           |
| <b>IMPLEMENTASI BONUS.....</b>                                           | <b>35</b> |
| 5.1. Structural Similarity Index (SSIM).....                             | 35        |
| 5.2. Target Kompresi.....                                                | 36        |
| 5.3. GIF.....                                                            | 36        |
| <b>BAB VI</b>                                                            |           |
| <b>EKSPERIMENT DAN ANALISIS.....</b>                                     | <b>37</b> |
| 6.1. Pengujian Program.....                                              | 37        |
| 6.2. Analisis Kompleksitas Algoritma Divide and Conquer Quadtree.....    | 40        |
| 6.2.1. Analisis Kompleksitas Pembuatan Quadtree.....                     | 40        |
| 6.2.2. Analisis Kompleksitas Algoritma Kompresi Berbasis Threshold.....  | 41        |
| 6.2.3. Analisis Kompleksitas Algoritma Kompresi Berbasis Percentage..... | 42        |
| 6.2.3. Analisis Kompleksitas Pembuatan GIF.....                          | 43        |
| 6.2.3. Analisis Kompleksitas Keseluruhan.....                            | 44        |
| <b>LAMPIRAN.....</b>                                                     | <b>45</b> |
| Checklist Spesifikasi Program.....                                       | 45        |
| Pembagian Tugas.....                                                     | 45        |
| Repository.....                                                          | 45        |
| <b>DAFTAR PUSTAKA.....</b>                                               | <b>47</b> |

## **DAFTAR GAMBAR**

|                                                              |   |
|--------------------------------------------------------------|---|
| Gambar 1.1. Quadtree dalam kompresi gambar                   | 5 |
| Gambar 2.1. Kompleksitas waktu algoritma divide and conquer. | 7 |

## BAB I

### DESKRIPSI MASALAH



Gambar 1.1. *Quadtree* dalam kompresi gambar

(Sumber: <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>)

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

Dalam tugas ini, program dituliskan dalam bahasa **C#** (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.
6. [INPUT] alamat absolut gambar hasil kompresi.
7. [INPUT] alamat absolut GIF.
8. [INPUT] waktu eksekusi.
9. [OUTPUT] ukuran (memori) gambar sebelum dikompresi.
10. [OUTPUT] ukuran (memori) gambar setelah dikompresi.
11. [OUTPUT] persentase kompresi.
12. [OUTPUT] kedalaman pohon (*quadtree*).
13. [OUTPUT] banyak simpul (*node*) pada pohon.
14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.
15. [OUTPUT] GIF proses kompresi pada lamata yang sudah ditentukan.

## BAB II

### TEORI SINGKAT

#### 2.1. Divide and Conquer

Algoritma divide and conquer adalah teknik penyelesaian masalah yang dilakukan melalui tiga tahap utama, yaitu *divide* (membagi), *conquer* (menaklukkan), dan *combine* (menggabungkan). Pada tahap divide, masalah awal akan dipecah menjadi beberapa sub-masalah yang lebih kecil, tetapi tetap memiliki karakteristik serupa dengan persoalan awal. Idealnya, sub-masalah yang terbentuk berukuran relatif seimbang.

Selanjutnya, tahap conquer adalah menyelesaikan setiap sub-masalah tersebut. Apabila sub-masalah telah cukup kecil, maka dapat langsung diselesaikan secara langsung; namun, jika ukurannya masih besar, penyelesaiannya akan dilakukan secara rekursif dengan menerapkan kembali algoritma yang sama. Terakhir, pada tahap combine, solusi-solusi dari setiap sub-masalah yang telah diselesaikan akan digabungkan untuk menghasilkan solusi dari masalah awal secara utuh.

Metode divide and conquer sangat cocok diterapkan pada persoalan yang dapat dipilah menjadi bagian-bagian kecil yang mirip satu sama lain. Meskipun mungkin terkesan rumit pada awalnya, algoritma ini justru menawarkan penyelesaian yang lebih efisien dan efektif, karena memecah persoalan besar menjadi sub-masalah kecil yang lebih mudah diselesaikan. Dengan keunggulan tersebut, algoritma ini dapat diterapkan secara luas, seperti pada pemrosesan paralel, pengurutan data (*sorting*), maupun pencarian (*searching*). Secara umum, kompleksitas waktu dari algoritma *divide and conquer* dapat dirumuskan sebagai berikut:

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \dots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

Gambar 2.1. Kompleksitas waktu algoritma *divide and conquer*.

(Sumber: [Website Pak Rinaldi](#))

dengan:

- ✓  $T(n)$  : kompleksitas waktu penyelesaian persoalan P yang berukuran n
- ✓  $g(n)$  : kompleksitas waktu untuk **Conquer** jika n sudah berukuran kecil
- ✓  $T(n_1) + T(n_2) + \dots + T(n_r)$  : kompleksitas waktu untuk memproses setiap sub-persoalan
- ✓  $f(n)$  : kompleksitas waktu untuk **Combine** solusi dari setiap sub-persoalan
- \* Tahap **Divide** dilakukan dengan kompleksitas O(1).

## 2.2. *Quadtree*

Quadtree adalah struktur data berbentuk pohon (*tree*) yang umumnya digunakan untuk merepresentasikan data spasial dua dimensi secara efisien. Nama "Quadtree" berasal dari konsep pembagian ruang menjadi empat bagian (*quad*) secara berulang-ulang. Setiap simpul (*node*) dalam *Quadtree* dapat memiliki empat anak (*child nodes*), masing-masing merepresentasikan satu kuadran dari area induknya. Proses pembentukan *Quadtree* mirip dengan pendekatan *divide and conquer*, dimana ruang dua dimensi akan **dibagi menjadi empat** bagian yang lebih kecil dengan ukuran relatif sama. Jika setiap bagian tersebut masih dianggap kompleks atau belum mencapai kriteria tertentu, proses pembagian akan terus dilakukan secara rekursif hingga mencapai tingkat kedalaman tertentu atau kriteria tertentu terpenuhi.

Dalam konteks mengompres gambar, *Quadtree* dimanfaatkan sebagai struktur data yang memiliki efisiensi tinggi dalam menyimpan informasi gambar dengan pendekatan berbasis *divide and conquer*. Ide utamanya adalah membagi gambar secara rekursif, kemudian melakukan evaluasi apakah setiap kuadran tersebut sudah cukup homogen, misalnya memiliki warna atau intensitas piksel yang relatif sama. Jika suatu area masih dianggap terlalu kompleks atau heterogen, area tersebut akan terus dibagi lebih lanjut hingga mencapai area-area kecil dengan tingkat homogenitas yang diinginkan.

Setelah proses pembagian selesai, setiap wilayah kecil yang dianggap homogen akan direpresentasikan dengan informasi yang lebih sederhana. Dengan demikian, kita hanya perlu menyimpan informasi yang disederhanakan dari setiap wilayah yang telah direpresentasikan oleh simpul-simpul daun (*leaf nodes*) pada Quadtree. Hal ini mengurangi ukuran data yang dibutuhkan untuk menyimpan gambar secara signifikan. Dengan menggunakan struktur Quadtree, gambar dapat disimpan secara lebih ringkas, efisien, dan adaptif terhadap detail gambar. Daerah yang memiliki detail tinggi akan mendapatkan pembagian yang lebih dalam (lebih detail), sementara area yang homogen akan direpresentasikan dengan pembagian yang lebih dangkal. Pendekatan ini memberikan hasil kompresi yang optimal, menjaga kualitas visual gambar sekaligus mengurangi ukuran data secara efektif.

## **BAB III**

### **METODE PENYELESAIAN**

Algoritma *divide and conquer* dapat diterapkan dalam proses kompresi gambar dengan metode *quadtree*. Jika gambar direpresentasikan sebagai ‘*2 dimensional array of vectors*’ (*RGB*), maka kita dapat memulai dari pembagian (*divide*) bagian array hingga *depth* tertentu. Lalu bagian yang relatif memiliki warna (*RGB vector’s value*) yang sama akan digabungkan kembali menjadi satu warna. Untuk lebih detailnya sebagai berikut:

#### **3.1. *Divide* (Pembagian)**

Pada tahap ini, gambar awal direpresentasikan sebagai sebuah array dua dimensi yang terdiri dari piksel-piksel, masing-masing memiliki nilai vektor warna *RGB*. Gambar dibagi secara rekursif menjadi empat wilayah kuadran dengan ukuran yang sama hingga mencapai tingkat kedalaman tertentu (*depth*) atau hingga ukuran tertentu. Setiap kuadran yang terbentuk akan terus dibagi jika kuadran tersebut memiliki piksel dengan warna yang cukup berbeda satu sama lain (tidak homogen). Sebaliknya, jika kuadran sudah dianggap homogen (misalnya perbedaan warna antara piksel cukup kecil), pembagian dapat dihentikan untuk wilayah tersebut.

Sebagai contoh sederhana, apabila gambar awal berukuran 512x512 piksel, gambar tersebut akan dibagi menjadi empat kuadran 256x256 piksel, kemudian masing-masing kuadran ini dibagi lagi menjadi kuadran yang lebih kecil, misalnya 128x128 piksel, dan proses ini berlangsung secara rekursif hingga kondisi tertentu terpenuhi.

#### **3.2. *Conquer/Solve* (Penyelesaian)**

Pada tahap *Conquer*, masing-masing wilayah kecil hasil pembagian tersebut dievaluasi tingkat homogenitas warnanya. Evaluasi ini dilakukan dengan menghitung kesalahan atau perbedaan warna antara piksel-piksel dalam wilayah tersebut, misalnya dengan menghitung perbedaan maksimum antar nilai *RGB* dalam blok atau menggunakan rata-rata warna di wilayah tersebut.

Hasil evaluasi ini menentukan apakah suatu wilayah dianggap cukup homogen atau tidak. Bila perbedaan warna atau tingkat kesalahannya berada di bawah batas (*threshold*) tertentu, wilayah tersebut dianggap homogen. Sebaliknya, jika perbedaannya masih terlalu besar, wilayah ini harus terus dibagi lebih lanjut pada tahap sebelumnya hingga memenuhi kriteria homogenitas.

#### **3.3. *Combine* (Penggabungan)**

Pada tahap terakhir, wilayah-wilayah kecil yang telah dianggap homogen akan digabungkan kembali dan direpresentasikan dengan informasi yang disederhanakan. Dalam program ini, suatu blok

wilayah homogen akan direpresentasikan hanya dengan satu nilai warna tunggal (rata-rata RGB dari seluruh piksel dalam blok tersebut).

Wilayah-wilayah yang heterogen sebelumnya sudah terbagi lebih lanjut hingga mencapai blok-blok yang homogen, sehingga gambar akhir akan memiliki sejumlah wilayah yang tiap-tiapnya diwakili oleh nilai warna tunggal. Dengan demikian, ukuran gambar secara keseluruhan menjadi jauh lebih kecil karena informasi yang disimpan berkurang signifikan.

## **BAB IV**

### **IMPLEMENTASI**

#### **4.1. Implementasi Program**

Implementasi program dibagi menjadi beberapa sumber kode. Masing-masing sumber kode memiliki peran khusus untuk penyelesaian masalah kompresi gambar. Pembagian sumber kode dilakukan untuk menyederhanakan masalah agar dapat lebih mudah dipahami. Berikut adalah kode sumber yang ada dalam implementasi program.

##### 4.1.1. Error Calculation

Kode sumber ini berisi kelas-kelas untuk menghitung error yang menentukan apakah bagian gambar akan dikompres atau tidak. Perhitungan error yang didukung dalam kode sumber ini adalah Variance, Mean Absolute Difference, Max Pixel Difference, Entropy, dan SSIM.

###### 4.1.1.1. Abstract Class: ErrorCalculator

| <b>Nama</b>         | <b>Tipe</b>                  | <b>Keterangan</b>                                                                                 |
|---------------------|------------------------------|---------------------------------------------------------------------------------------------------|
| image               | Image<Rgba32><br>(Protected) | Referensi ke gambar yang akan dihitung error-nya; diisi melewati LoadImage(...)                   |
| LoadImage(...)      | void                         | Menginisialisasi/mereset image; dapat di-override untuk mengosongkan cache internal sebelum load. |
| Name                | string (abstract)            | Nama metode kalkulasi error (implementasi di subclass)                                            |
| CalculateError(...) | double (abstract)            | Hitung dan kembalikan nilai error untuk Region2Int region                                         |

###### 4.1.1.2. MaxPixelDifferenceCalculator

| <b>Nama</b>         | <b>Returns</b> | <b>Parameter</b>      | <b>Keterangan</b>                                                             |
|---------------------|----------------|-----------------------|-------------------------------------------------------------------------------|
| Name                | string         | —                     | “Max Pixel Difference”                                                        |
| CalculateError(...) | double         | region:<br>Region2Int | Selisih maksimum per-channel (R,G,B) dibagi (256*3), menghasilkan nilai [0-1] |

###### 4.1.1.3. VarianceCalculator

| <b>Nama</b>         | <b>Returns</b> | <b>Parameter</b>      | <b>Keterangan</b>                                      |
|---------------------|----------------|-----------------------|--------------------------------------------------------|
| Name                | string         | —                     | “Variance”                                             |
| CalculateError(...) | double         | region:<br>Region2Int | Variansi rata-rata per-channel, dinormalisasi terhadap |

|  |  |  |                                                      |
|--|--|--|------------------------------------------------------|
|  |  |  | $\maxVariance = 127.5^2$ , lalu dirata-rata ke [0-1] |
|--|--|--|------------------------------------------------------|

#### 4.1.1.4. MeanAbsoluteDeviationCalculator

| Nama                | Returns | Parameter             | Keterangan                                                                                          |
|---------------------|---------|-----------------------|-----------------------------------------------------------------------------------------------------|
| Name                | string  | —                     | “Mean Absolute Deviation (MAD)”                                                                     |
| CalculateError(...) | double  | region:<br>Region2Int | Deviasi absolut rata-rata per-channel, dirata-rata dan dinormalisasi terhadap 127.5, dibatasi [0-1] |

#### 4.1.1.5. EntropyCalculator

| Nama                          | Returns | Parameter                       | Keterangan                                                                                                  |
|-------------------------------|---------|---------------------------------|-------------------------------------------------------------------------------------------------------------|
| Name                          | string  | —                               | “Entropy”                                                                                                   |
| CalculateError(...)           | double  | region:<br>Region2Int           | Entropi Shannon per-channel<br>$(-\sum p \log_2 p)$ , rata-rata dibagi 8 (max entropy 8 bit), dibatas [0-1] |
| CalculateChannel Entropy(...) | double  | freq: int[],<br>totalCount: int | Hitung entropi untuk satu channel berdasarkan histogram frekuensi                                           |

#### 4.1.1.5. SSIMCalculator

| Nama                | Returns             | Parameter                                 | Keterangan                                                                                                                                  |
|---------------------|---------------------|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| Name                | string              | —                                         | “SSIM” (self-comparison dalam satu region)                                                                                                  |
| CalculateError(...) | double              | region:<br>Region2Int                     | SSIM rata-rata per-channel: menggunakan konstanta $C_1$ , $C_2$ , menghitung mean, var, cov (self = var), lalu formula SSIM, dibatasi [0-1] |
| CalculateSSIM(...)  | double<br>(private) | muX, muY, varX,<br>varY, covXY, C1,<br>C2 | Formula inti SSIM: Dapat dilihat di bab implementasi bonus ( <a href="#">disini</a> )                                                       |

#### 4.1.2. Image Compressor

Kode sumber ini merupakan kode utama dari program. Kode sumber ini mengandung proses input, output, serta algoritma kompresi menggunakan threshold dan target kompresi. Kode sumber ini juga mengandung beberapa fungsi tambahan seperti fungsi normalisasi bagian dari gambar.

#### 4.1.2.1. Field dan Properti Publik

| Nama                      | Tipe              | Keterangan                                         |
|---------------------------|-------------------|----------------------------------------------------|
| availableErrorCalculators | ErrorCalculator[] | Daftar instansi error calculator yang dipilih user |
| framerate                 | integer (static)  | Frame delay untuk GIF (0.01 detik)                 |
| imagePath                 | string            | Path input gambar                                  |
| errorCalculator           | ErrorCalculator   | Kalkulator error yang dipilih                      |
| threshold                 | double            | Ambang error untuk metode threshold                |
| minBlockSize              | integer           | Ukuran minimum blok untuk subdivisi                |
| compressionTarget         | double            | Target kompresi (persentase)                       |
| outPath                   | string            | Path output gambar                                 |
| gifPath                   | string            | Path output GIF                                    |
| executionTime             | double            | Waktu eksekusi kompresi (ms)                       |
| originalSize              | long              | Ukuran file asli (bytes)                           |
| newSize                   | long              | Ukuran file hasil kompresi (bytes)                 |
| compressionPercentage     | double            | Persentase pengurangan ukuran                      |
| treeDepth                 | int               | Kedalaman quad-tree yang digunakan                 |
| treeNodes                 | int               | Jumlah node quad-tree yang dihasilkan              |

#### 4.1.2.2. Field Privat

| Nama        | Tipe           | Keterangan                                         |
|-------------|----------------|----------------------------------------------------|
| rawData     | byte[]?        | Buffer file input/output untuk penghitungan ukuran |
| sourceImage | Image<Rgba32>? | Gambar sumber sebagai ImageSharp                   |
| outImage    | Image<Rgba32>? | Gambar hasil kompresi yang di update per-region    |
| gif         | Image<Rgba32>? | Objek GIF yang diisi frame per frame               |
| encode      | IImageEncoder? | Encoder sesuai format gambar input                 |
| tree        | QuadTree?      | QuadTree yang digunakan pada mode persentase       |

#### 4.1.2.3. Konstruktur

Menginisialisasi field dengan nilai-nilai default.

#### 4.1.2.4. Metode Publik

| Nama | Returns | Parameter     | Keterangan                                                                                                                                                          |
|------|---------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Main | void    | string[] args | Titik masuk program: interaksi CLI untuk input/output, memilih metode error, threshold, block size, target kompresi, dan menjalankan Run()                          |
| Run  | void    | —             | Proses utama: baca file, load image, setup GIF, load errorCalculator, pilih mode (threshold atau percentage), panggil metode kompresi, simpan output, hitung metrik |

#### 4.1.2.5. Metode Privat

| Nama                  | Returns | Parameter                     | Keterangan                                                                                                                     |
|-----------------------|---------|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| CompressBy-Threshold  | void    | —                             | Gunakan rekursi untuk menelusuri QuadTree, menghitung error, dan normalisasi berdasarkan <i>threshold</i>                      |
| RecursiveCompre-ss    | void    | QuadTree.Node node, int depth | Subdivisi rekursif berdasarkan threshold: jika error < threshold, normalisasi region; else jika region besar, subdivisi lagi   |
| CompressBy-Percentage | void    | —                             | Gunakan QuadTree untuk iteratif pop region dengan error terkecil hingga mencapai target ukuran file                            |
| GenerateGIF           | void    | —                             | Membuat GIF menggunakan QuadTree yang dibentuk selama proses kompresi                                                          |
| NormalizeRegion       | void    | Region2Int region             | Hitung rata-rata warna pada region di sourceImage, lalu apply warna rata-rata ke outImage; tambahkan frame GIF secara periodik |

### 4.1.3. Tree

Kode sumber ini berisi kelas Quadtree yang digunakan dalam kompresi menggunakan target kompresi. Quadtree menggambarkan struktur dan urutan pengkompresian yang digunakan dalam algoritma kompresi menggunakan target kompresi.

#### 4.1.3.1. Atribut Publik

| Nama      | Tipe    | Keterangan                                                |
|-----------|---------|-----------------------------------------------------------|
| treeNodes | integer | Jumlah total node yang dibuat selama konstruksi quad-tree |
| treeDepth | integer | Kedalaman maksimum quad-tree                              |

|             |         |                                                |
|-------------|---------|------------------------------------------------|
| leavesCount | integer | Jumlah daun saat ini (same as leafNodes.count) |
|-------------|---------|------------------------------------------------|

#### 4.1.3.2. Konstruktor

| Parameter       | Tipe            | Keterangan                                                            |
|-----------------|-----------------|-----------------------------------------------------------------------|
| source          | Image<Rgba32>   | Gambar sumber yang akan dibagi menjadi quad-tree                      |
| minBlockSize    | integer         | Ukuran minimum gambar ( $pixel^2$ ) sebelum blok dihentikan pembagian |
| errorCalculator | ErrorCalculator | Objek untuk menghitung nilai error pada region                        |

#### 4.1.3.3. Alur konstruksi

1. Buat **rootNode** dengan region seluas seluruh gambar.
2. Gunakan **LinkedList<Node>** sebagai antrian “daun sementara” (**tempLeaves**).
3. Selama ada node di antrian, jika region cukup besar (**area  $\geq 4 * minBlockSize$**  dan **size.x,y > 1**), bagi menjadi 4 anak dan tambahkan ke antrian; jika tidak, biarkan sebagai daun.
4. Hitung **treeDepth** dengan menelusuri parent dari daun terakhir.
5. Hitung **error** untuk setiap daun dengan **errorCalculator.CalculateError(...)**.
6. Panggil **SortLeaves()** untuk membangun min-heap berdasarkan error.

#### 4.1.3.4. Metode

| Publik                        |             |            |                                                                                                                                                                            |
|-------------------------------|-------------|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nama                          | Tipe        | Parameter  | Keterangan                                                                                                                                                                 |
| Pop()                         | ImageRegion | —          | Mengeluarkan (pop) daun dengan error terkecil, memperbaiki heap, dan jika parent kehilangan semua anak, masukkan kembali parent sebagai daun baru dengan error terbarunya. |
| Private                       |             |            |                                                                                                                                                                            |
| SortLeaves()                  | void        | —          | Membangun min-heap dari leafNodes dengan menyisipkan setiap elemen ke posisi yang benar (bottom-up).                                                                       |
| InsertLeafNode<br>(Node node) | void        | node: Node | Menambahkan node ke leafNodes dan memperbaiki heap (shift-up) berdasarkan content.error.                                                                                   |

#### 4.1.3.5 Private Field

| Nama | Tipe | Keterangan |
|------|------|------------|
|------|------|------------|

|                 |                 |                                                                 |
|-----------------|-----------------|-----------------------------------------------------------------|
| sourceImage     | Imgae(Rgba32)   | Referensi ke gambar sumber                                      |
| errorCalculator | ErrorCalculator | Objek untuk perhitungan error                                   |
| rootNode        | Node            | Node akar quad-tree                                             |
| leafNodes       | List<Nodes>     | Daftar node daun yang diatur sebagai min-heap berdasarkan error |
| _treeNodes      | integer         | Counter untuk total node yang dibuat                            |
| _treeDepth      | integer         | Counter untuk kedalaman tree                                    |

#### 4.1.3.6 Nested Class; Node

| Nama Field | Tipe        | Keterangan                                                                          |
|------------|-------------|-------------------------------------------------------------------------------------|
| content    | ImageRegion | Region dan nilai error yang dipegang oleh node                                      |
| compressed | bool        | Apakah node ini sudah terkompresi?                                                  |
| parent     | Node?       | Referensi ke parent node (root = null)                                              |
| children   | Node?[]     | Array panjang 4 untuk keempat kuadran; null jika tidak ada anak di kuadran tersebut |

#### 4.1.4. Util

Kode sumber ini berisi kelas-kelas pembantu yang mempermudah perhitungan pada beberapa bagian program. Kelas-kelas yang terdapat dalam kode sumber ini berurusan dengan menyimpan dan menghitung koordinat pada gambar.

##### 4.1.4.1. Atribut Vector2Int

| Nama | Tipe    | Keterangan            |
|------|---------|-----------------------|
| x    | integer | Absis (koordinat x)   |
| y    | integer | Ordinat (koordinat y) |

##### 4.1.4.2. Operator Vector2Int

| Operator | Returns    | Keterangan                                |
|----------|------------|-------------------------------------------|
| a+b      | Vector2Int | Penjumlahan vektor                        |
| a-b      | Vector2Int | Pengurangan vektor                        |
| v*s      | Vector2Int | Perkalian skalar vektor (setiap v kali s) |
| s*v      | Vector2Int | Perkalian skalar vektor (sama dengan v*s) |
| v/s      | Vector2Int | Pembagian skalar vektor (pembulatan int)  |

#### 4.1.4.3. Atribut Region2Int

| Nama    | Tipe       | Keterangan                                     |
|---------|------------|------------------------------------------------|
| start   | Vector2Int | Koordinat pojok kiri atas (inklusif) wilayah   |
| end     | Vector2Int | Koordinat pojok kanan bawah (inklusif) wilayah |
| size    | Vector2Int | Ukuran wilayah = (end-start) + (1,1)           |
| area    | integer    | Luas wilayah = (size.x * size.y)               |
| isValid | boolean    | Validitas wilayah : True if start ≤ end        |

#### 4.1.4.3. Atribut ImageRegion

| Nama   | Tipe       | Keterangan                    |
|--------|------------|-------------------------------|
| region | Region2Int | Wilayah gambar yang diukur    |
| error  | double     | Galat untuk kompresi/analisis |

## 4.2. Kode Sumber

### 4.2.1. Error Calculation

```
namespace ImageCompressor.ErrorCalculation;

using System.Diagnostics.CodeAnalysis;
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.PixelFormats;
using Util;

public abstract class ErrorCalculator
{
    protected Image<Rgba32>? image = null;

    // This function should be overloaded when implementing a cache to reset the internal cache
    public virtual void LoadImage(Image<Rgba32> image)
    {
        this.image = image;
    }

    public abstract string Name { get; }
    public abstract double CalculateError(Region2Int region);
}

public class MaxPixelDifferenceCalculator : ErrorCalculator
{
    public override string Name { get => "Max Pixel Difference"; }

    public override void LoadImage(Image<Rgba32> image)
    {
        base.LoadImage(image);
    }

    public override double CalculateError(Region2Int region)
```

```

{
    byte minR = byte.MaxValue;
    byte minG = byte.MaxValue;
    byte minB = byte.MaxValue;

    byte maxR = byte.MinValue;
    byte maxG = byte.MinValue;
    byte maxB = byte.MinValue;

    for (int j = region.start.y; j <= region.end.y; j++)
    {
        for (int i = region.start.x; i < region.end.x; i++)
        {
            Rgba32 c = image![i, j];

            if (c.R > maxR) maxR = c.R;
            if (c.G > maxG) maxG = c.G;
            if (c.B > maxB) maxB = c.B;

            if (c.R < minR) minR = c.R;
            if (c.G < minG) minG = c.G;
            if (c.B < minB) minB = c.B;
        }
    }

    return (maxR + maxG + maxB - minR - minG - minB) / (255.0d * 3.0d);
}

public class VarianceCalculator : ErrorCalculator
{
    public override string Name { get => "Variance"; }

    public override double CalculateError(Region2Int region)
    {
        if (image == null)
            return 0;

        long sumR = 0, sumG = 0, sumB = 0;
        int count = 0;

        for (int j = region.start.y; j <= region.end.y; j++)
        {
            for (int i = region.start.x; i <= region.end.x; i++)
            {
                Rgba32 c = image![i, j];
                sumR += c.R;
                sumG += c.G;
                sumB += c.B;
                count++;
            }
        }

        if (count == 0)
            return 0;

        double meanR = (double)sumR / count;
        double meanG = (double)sumG / count;
        double meanB = (double)sumB / count;
    }
}

```

```

double sqDiffR = 0, sqDiffG = 0, sqDiffB = 0;
for (int j = region.start.y; j <= region.end.y; j++)
{
    for (int i = region.start.x; i <= region.end.x; i++)
    {
        Rgba32 c = image![i, j];
        sqDiffR += Math.Pow(c.R - meanR, 2);
        sqDiffG += Math.Pow(c.G - meanG, 2);
        sqDiffB += Math.Pow(c.B - meanB, 2);
    }
}

// Compute average variance for each color channel
double varianceR = sqDiffR / count;
double varianceG = sqDiffG / count;
double varianceB = sqDiffB / count;

// Max deviation possible is 127.5 and the variance is 127.5^2 = 16256.25.
double maxVariance = 16256.25;

double normalizedVarianceR = varianceR / maxVariance;
double normalizedVarianceG = varianceG / maxVariance;
double normalizedVarianceB = varianceB / maxVariance;

double normalizedVariance = (normalizedVarianceR + normalizedVarianceG + normalizedVarianceB) /
3.0;

return Math.Min(1.0, Math.Max(0.0, normalizedVariance));
}
}

public class MeanAbsoluteDeviationCalculator : ErrorCalculator
{
    public override string Name { get => "Mean Absolute Deviation (MAD)"; }

    public override double CalculateError(Region2Int region)
    {
        if (image == null) return 0;

        long sumR = 0, sumG = 0, sumB = 0;
        int count = 0;
        for (int j = region.start.y; j <= region.end.y; j++)
        {
            for (int i = region.start.x; i <= region.end.x; i++)
            {
                Rgba32 c = image[i, j];
                sumR += c.R;
                sumG += c.G;
                sumB += c.B;
                count++;
            }
        }

        if (count == 0) return 0;

        double meanR = (double)sumR / count;
        double meanG = (double)sumG / count;
        double meanB = (double)sumB / count;
    }
}

```

```

        double absDiffR = 0, absDiffG = 0, absDiffB = 0;
        for (int j = region.start.y; j <= region.end.y; j++)
        {
            for (int i = region.start.x; i <= region.end.x; i++)
            {
                Rgba32 c = image[i, j];
                absDiffR += Math.Abs(c.R - meanR);
                absDiffG += Math.Abs(c.G - meanG);
                absDiffB += Math.Abs(c.B - meanB);
            }
        }

        double madR = absDiffR / count;
        double madG = absDiffG / count;
        double madB = absDiffB / count;

        double madRGB = (madR + madG + madB) / 3.0;

        double normalized = madRGB / 127.5;

        if (normalized < 0) normalized = 0;
        if (normalized > 1) normalized = 1;

        return normalized;
    }
}

public class EntropyCalculator : ErrorCalculator
{
    public override string Name { get => "Entropy"; }

    public override double CalculateError(Region2Int region)
    {
        if (image == null) return 0;

        int[] freqR = new int[256];
        int[] freqG = new int[256];
        int[] freqB = new int[256];

        int count = 0;
        for (int j = region.start.y; j <= region.end.y; j++)
        {
            for (int i = region.start.x; i <= region.end.x; i++)
            {
                Rgba32 c = image[i, j];
                freqR[c.R]++;
                freqG[c.G]++;
                freqB[c.B]++;
                count++;
            }
        }

        if (count == 0) return 0;

        double hR = CalculateChannelEntropy(freqR, count);
        double hG = CalculateChannelEntropy(freqG, count);
        double hB = CalculateChannelEntropy(freqB, count);
    }
}

```

```

        double hRGB = (hR + hG + hB) / 3.0;

        double normalized = hRGB / 8.0;

        if (normalized < 0) normalized = 0;
        if (normalized > 1) normalized = 1;

        return normalized;
    }

    private double CalculateChannelEntropy(int[] freq, int totalCount)
    {
        double h = 0.0;
        for (int i = 0; i < 256; i++)
        {
            if (freq[i] == 0) continue;

            double p = (double)freq[i] / totalCount;
            // sum= -p * log2(p)
            h += -p * Math.Log2(p);
        }
        return h;
    }

    // Katanya seluruh Error Calculation disini, jadi yowes ini juga disini, help code nya panjang T^T
    public class SSIMCalculator : ErrorCalculator
    {
        public override string Name { get => "SSIM"; }

        public override double CalculateError(Region2Int region)
        {
            // SSIM disini berarti self-comparison
            if (image == null) return 0;

            // Typical SSIM constants for 8-bit images
            const double L = 255.0;    // max pixel value
            const double k1 = 0.01;
            const double k2 = 0.03;
            double C1 = (k1 * L) * (k1 * L); // (0.01*255)^2 ~ 6.5025
            double C2 = (k2 * L) * (k2 * L); // (0.03*255)^2 ~ 58.5225

            int count = 0;
            long sumR = 0, sumG = 0, sumB = 0;
            for (int j = region.start.y; j <= region.end.y; j++)
            {
                for (int i = region.start.x; i <= region.end.x; i++)
                {
                    Rgba32 c = image[i, j];
                    sumR += c.R;
                    sumG += c.G;
                    sumB += c.B;
                    count++;
                }
            }
            if (count == 0) return 0;

            double meanR = (double)sumR / count;
            double meanG = (double)sumG / count;
            double meanB = (double)sumB / count;
        }
    }
}

```

```

double meanB = (double)sumB / count;

// Compute variances (for single-image self-comparison, cross-cov ~ var)
double varR = 0, varG = 0, varB = 0;
for (int j = region.start.y; j <= region.end.y; j++)
{
    for (int i = region.start.x; i <= region.end.x; i++)
    {
        Rgba32 c = image[i, j];
        varR += Math.Pow(c.R - meanR, 2);
        varG += Math.Pow(c.G - meanG, 2);
        varB += Math.Pow(c.B - meanB, 2);
    }
}
varR /= count;
varG /= count;
varB /= count;

// for this (self comparison) cross-cov = var.
double covR = varR, covG = varG, covB = varB;

double ssimR = ComputeSSIM(meanR, meanR, varR, 0, 0, C1, C2);
double ssimG = ComputeSSIM(meanG, meanG, varG, 0, 0, C1, C2);
double ssimB = ComputeSSIM(meanB, meanB, varB, 0, 0, C1, C2);

double ssimRGB = ssimR * 0.299f + ssimG * 0.587f + ssimB * 0.114f;

if (ssimRGB < 0) ssimRGB = 0;
if (ssimRGB > 1) ssimRGB = 1;

return 1 - ssimRGB;
}

private double ComputeSSIM(
    double muX, double muY,
    double varX, double varY,
    double covXY,
    double C1, double C2
)
{
    double numerator = (2.0 * muX * muY + C1) * (2.0 * covXY + C2);
    double denominator = (muX * muX + muY * muY + C1) * (varX + varY + C2);

    if (denominator == 0.0) return 1.0; // mudah-mudahan gak perlu, but just in case

    return numerator / denominator;
}
}

```

#### 4.2.2. Image Compressor

```

namespace ImageCompressor;

using SixLabors.ImageSharp;
using Tree;
using ErrorCalculation;

```

```

using Util;
using SixLabors.ImageSharp.PixelFormats;
using SixLabors.ImageSharp.Formats;
using System.Diagnostics;
using SixLabors.ImageSharp.Processing;
using SixLabors.ImageSharp.Drawing.Processing;

class ImageCompressor
{
    static void Main(string[] args)
    {
        ImageCompressor compressor = new ImageCompressor();

        Console.WriteLine("Image Compressor v1.0");

        do
        {
            Console.Write("Enter an absolute image path: ");
            compressor.imagePath = Console.ReadLine() ?? "";
        }
        while (!File.Exists(compressor.imagePath));

        do
        {
            Console.WriteLine("Choose an error calculation method.");
            Console.WriteLine("Possible values are:");

            for (int i = 0; i < availableErrorCalculators.Length; i++)
            {
                Console.WriteLine("> " + availableErrorCalculators[i].Name);
            }

            string selection = Console.ReadLine() ?? "";
            bool validSelection = false;

            for (int i = 0; i < availableErrorCalculators.Length; i++)
            {
                if (selection == availableErrorCalculators[i].Name)
                {
                    compressor.errorCalculator = availableErrorCalculators[i];
                    validSelection = true;
                    break;
                }
            }

            if (validSelection) break;
        }
        while (true);

        do
        {
            Console.Write("Enter desired threshold: ");

            string rawInput = Console.ReadLine() ?? "";

```

```

if (!double.TryParse(rawInput, out compressor.threshold))
{
    continue;
}

if (compressor.threshold < 0d || compressor.threshold > 1d)
{
    continue;
}

break;
}
while (true);

do
{
    Console.Write("Enter minimum block size: ");

    string rawInput = Console.ReadLine() ?? "";

    if (!int.TryParse(rawInput, out compressor.minBlockSize))
    {
        continue;
    }

    if (compressor.minBlockSize < 1)
    {
        continue;
    }

    break;
}
while (true);

do
{
    Console.Write("Enter desired compression rate: ");

    string rawInput = Console.ReadLine() ?? "";

    if (!double.TryParse(rawInput, out compressor.compressionTarget))
    {
        continue;
    }

    if (compressor.compressionTarget < 0d || compressor.compressionTarget > 1d)
    {
        continue;
    }

    break;
}
while (true);

```

```

do
{
    Console.WriteLine("Enter an absolute output path (should have the same extension as input
file):");
    compressor.outPath = Console.ReadLine() ?? "";
}
while (compressor.outPath == "");

do
{
    Console.WriteLine("Enter an absolute output path for GIF:");
    compressor.gifPath = Console.ReadLine() ?? "";
}
while (compressor.gifPath == "");

Console.WriteLine("\nCompressing. Please Wait...\n");

compressor.Run();

Console.WriteLine("Compression completed in " + compressor.executionTime.ToString() + "
milliseconds");
Console.WriteLine("Original size: " + compressor.originalSize.ToString() +" bytes");
Console.WriteLine("Compressed size: " + compressor.newSize.ToString() + " bytes");
Console.WriteLine("Compression percentage: " +
compressor.compressionPercentage.ToString() + "%");
Console.WriteLine("Quadtree max depth: " + compressor.treeDepth.ToString());
Console.WriteLine("Quadtree nodes: " + compressor.treeNodes.ToString());
}

static ErrorCalculator[] availableErrorCalculators = {
    new MaxPixelDifferenceCalculator(),
    new VarianceCalculator(),
    new MeanAbsoluteDeviationCalculator(),
    new EntropyCalculator(),
    new SSIMCalculator()
};

static int framerate = 20;

public string imagePath;
public ErrorCalculator errorCalculator;
public double threshold;
public int minBlockSize;
public double compressionTarget;
public string outPath;
public string gifPath;

public double executionTime = 0;
public long originalSize = 0;
public long newSize = 0;
public double compressionPercentage = 0;
public int treeDepth = 0;
public int treeNodes = 0;

```

```

private byte[]? rawData = null;
private Image<Rgba32>? sourceImage = null;
public Image<Rgba32>? outImage = null;
public Image<Rgba32>? gif = null;
private IImageEncoder? encoder = null;
private QuadTree? tree = null;

public ImageCompressor()
{
    imagePath = "";
    errorCalculator = availableErrorCalculators[0];
    threshold = 0;
    minBlockSize = 0;
    compressionTarget = 0;
    outPath = "";
    gifPath = "";
}

public void Run()
{
    Stopwatch stopwatch = new Stopwatch();
    stopwatch.Start();

    rawData = File.ReadAllBytes(imagePath);
    originalSize = rawData.LongLength;

    sourceImage = Image.Load<Rgba32>(imagePath);
    outImage = Image.Load<Rgba32>(imagePath);

    encoder =
        outImage.Configuration.ImageFormatsManager.GetEncoder(outImage.Metadata.DecodedImageFormat!);

    errorCalculator.LoadImage(sourceImage);

    tree = new QuadTree(sourceImage!, minBlockSize, errorCalculator);

    if (compressionTarget <= 0.0d)
    {
        CompressByThreshold();
    }
    else
    {
        compressionTarget = 1 - compressionTarget;
        CompressByPercentage();
    }

    GenerateGif();

    outImage.Save(outPath, encoder);
    gif.SaveAsGif(gifPath);

    stopwatch.Stop();
}

```

```

executionTime = stopwatch.Elapsed.TotalMilliseconds;

rawData = File.ReadAllBytes(outPath);
newSize = rawData.LongLength;

compressionPercentage = (1d - (double) newSize / originalSize) * 100d;
}

private void CompressByThreshold()
{
    RecursiveCompress(tree!.rootNode, 1);
}

private void RecursiveCompress(QuadTree.Node? node, int depth)
{
    if (node == null) return;

    Region2Int region = node.content.region;

    treeNodes += 1;

    if (region.area <= 1) return;

    double error = errorCalculator.CalculateError(region);

    if (error < threshold)
    {
        NormalizeRegion(region, sourceImage!.Frames.RootFrame, outImage!.Frames.RootFrame);
        treeDepth = Math.Max(depth, treeDepth);
        node.compressed = true;
    }
    else if (region.area >= 4 * minBlockSize && region.size.x > 1 && region.size.y > 1)
    {
        RecursiveCompress(node.children[0], depth + 1);
        RecursiveCompress(node.children[1], depth + 1);
        RecursiveCompress(node.children[2], depth + 1);
        RecursiveCompress(node.children[3], depth + 1);
    }
    else
    {
        treeDepth = Math.Max(depth, treeDepth);
    }
}

private void CompressByPercentage()
{
    treeNodes = tree!.treeNodes;
    treeDepth = tree.treeDepth;

    int pixelsInImage = sourceImage!.Width * sourceImage!.Height;
    int uncompressedPixels = pixelsInImage;
    int targetSize = (int) (originalSize * compressionTarget);
}

```

```

MemoryStream imageStream = new((int) originalSize);
outImage!.Save(imageStream, encoder!);

while (imageStream.Length > targetSize)
{
    int targetUncompressedPixels = uncompressedPixels - (int) ((imageStream.Length - targetSize) * uncompressedPixels / imageStream.Length);

    //Console.WriteLine(uncompressedPixels.ToString() + " > " +
targetUncompressedPixels.ToString() + " | " + tree.leavesCount.ToString());

    do
    {
        if (tree.leavesCount <= 1) break;

        Region2Int region = tree.Pop().region;

        if (region.area < 4 * minBlockSize || region.size.x <= 1 || region.size.y <= 1)
        {
            uncompressedPixels -= region.area - 1;
        }
        else
        {
            uncompressedPixels -= 3;
        }

        NormalizeRegion(region, sourceImage!.Frames.RootFrame,
outImage!.Frames.RootFrame);
    }
    while (uncompressedPixels > targetUncompressedPixels);

    if (tree.leavesCount <= 1) break;

    imageStream.SetLength(0);
    outImage.Save(imageStream, encoder!);
}
}

private void GenerateGif()
{
    gif = sourceImage!.Clone();
    gif.Frames.RootFrame.Metadata.GetGifMetadata().FrameDelay = framerate;
    gif.Metadata.GetGifMetadata().RepeatCount = 0;

    List<QuadTree.Node> currentNodes = new List<QuadTree.Node>([tree!.rootNode]);
    List<QuadTree.Node> nextNodes = new List<QuadTree.Node>(currentNodes.Count * 4);
    int currentDepth = 0;

    while (currentNodes.Count > 0)
    {
        ImageFrame<Rgba32> currentFrame = gif.Frames[currentDepth];

        for (int i = 0; i < currentNodes.Count; i++)
        {

```

```

QuadTree.Node currentNode = currentNodes[i];

NormalizeRegion(currentNode.content.region, sourceImage.Frames.RootFrame,
currentFrame);

if (currentNode.compressed) continue;

for (int j = 0; j < currentNode.children.Length; j++)
{
    QuadTree.Node? childNode = currentNode.children[j];

    if (childNode is null) continue;

    nextNodes.Add(childNode);
}
}

currentNodes = nextNodes;
nextNodes = new List<QuadTree.Node>(currentNodes.Count * 4);
currentDepth++;

gif.Frames.AddFrame(gif.Frames[currentDepth - 1]);
gif.Frames[currentDepth].Metadata.GetGifMetadata().FrameDelay = framerate;
}
}

private void NormalizeRegion(Region2Int region, ImageFrame<Rgba32> sourceImage,
ImageFrame<Rgba32> outImage)
{
    if (region.area <= 1) return;

    int regionWidth = region.end.x - region.start.x + 1;
    int regionHeight = region.end.y - region.start.y + 1;
    int regionSize = regionWidth * regionHeight;

    int r = 0;
    int g = 0;
    int b = 0;

    for (int j = region.start.y; j <= region.end.y; j++)
    {
        for (int i = region.start.x; i <= region.end.x; i++)
        {
            Rgba32 pixel = sourceImage![i, j];
            r += pixel.R;
            g += pixel.G;
            b += pixel.B;
        }
    }

    r /= regionSize;
    g /= regionSize;
    b /= regionSize;
}

```

```

Rgba32 color = new Rgba32((byte) r, (byte) g, (byte) b);

for (int j = region.start.y; j <= region.end.y; j++)
{
    for (int i = region.start.x; i <= region.end.x; i++)
    {
        outImage![i, j] = color;
    }
}
}

```

#### 4.2.3. Tree

```

namespace ImageCompressor.Tree;

using ErrorCalculation;
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.PixelFormats;
using Util;

public class QuadTree
{
    public int treeNodes { get => _treeNodes; }
    public int treeDepth { get => _treeDepth; }
    public int leavesCount { get => leafNodes.Count; }
    public Node rootNode { get => _rootNode; }

    public QuadTree(Image<Rgba32> source, int minBlockSize, ErrorCalculator errorCalculator)
    {
        this.sourceImage = source;
        this.errorCalculator = errorCalculator;
        this._rootNode = new Node() { content = new ImageRegion(new Region2Int(0, 0,
source.Width - 1, source.Height - 1)) };

        this.leafNodes = new List<Node>(source.Width * source.Height / minBlockSize)
{ _rootNode };

        int current = 0;
        while (current < leafNodes.Count)
        {
            _treeNodes++;

            Node node = leafNodes[current];
            Region2Int currentRegion = node.content.region;

            if (currentRegion.area >= 4 * minBlockSize && currentRegion.size.x > 1 &&
currentRegion.size.y > 1)
            {
                node.children[0] = new Node()
                {

```

```

        parent = node,
        content = new ImageRegion(new Region2Int(currentRegion.start, currentRegion.start
+ currentRegion.size / 2 - new Vector2Int(1, 1)))
    };

    node.children[1] = new Node()
    {
        parent = node,
        content = new ImageRegion(new Region2Int(currentRegion.start + new
Vector2Int(currentRegion.size.x / 2, 0), currentRegion.start + new Vector2Int(currentRegion.size.x -
1, currentRegion.size.y / 2 - 1)))
    };

    node.children[2] = new Node()
    {
        parent = node,
        content = new ImageRegion(new Region2Int(currentRegion.start + new Vector2Int(0,
currentRegion.size.y / 2), currentRegion.start + new Vector2Int(currentRegion.size.x / 2 - 1,
currentRegion.size.y - 1)))
    };

    node.children[3] = new Node()
    {
        parent = node,
        content = new ImageRegion(new Region2Int(currentRegion.start + currentRegion.size
/ 2, currentRegion.end))
    };

    leafNodes[current] = node.children[0]!;
    leafNodes.Add(node.children[1]!);
    leafNodes.Add(node.children[2]!);
    leafNodes.Add(node.children[3]!);

    continue;
}

current++;
}

for (Node? node = leafNodes[leafNodes.Count - 1]; node is not null; node = node.parent)
{
    _treeDepth++;
}

for (int i = 0; i < leafNodes.Count; i++)
{
    leafNodes[i].content.error = errorCalculator.CalculateError(leafNodes[i].content.region);
}

SortLeaves();
}

public ImageRegion Pop()
{

```

```

Node poppedNode = leafNodes[0];

leafNodes[0] = leafNodes[leafNodes.Count - 1];
leafNodes.RemoveAt(leafNodes.Count - 1);

for (int i = 2; i <= leafNodes.Count; i *= 2)
{
    if (i < leafNodes.Count && leafNodes[i].content.error < leafNodes[i - 1].content.error)
    {
        i++;
    }

    if (leafNodes[i / 2 - 1].content.error > leafNodes[i - 1].content.error)
    {
        (leafNodes[i / 2 - 1], leafNodes[i - 1]) = (leafNodes[i - 1], leafNodes[i / 2 - 1]);
    }
    else
    {
        break;
    }
}

poppedNode.compressed = true;

if (poppedNode.parent is null)
{
    return poppedNode.content;
}

Node parentNode = poppedNode.parent;
bool childrenless = true;

for (int i = 0; i < 4; i++)
{
    if (!parentNode.children[i]!.compressed)
    {
        childrenless = false;
    }
}

if (childrenless)
{
    parentNode.content.error = errorCalculator.CalculateError(parentNode.content.region);
    InsertLeafNode(parentNode);
}

return poppedNode.content;
}

private Image<Rgba32> sourceImage;
private ErrorCalculator errorCalculator;
private Node _rootNode;
private List<Node> leafNodes;
private int _treeNodes = 0;

```

```

private int _treeDepth = 0;

private void SortLeaves()
{
    for (int i = leafNodes.Count - 1; i > 0; i--)
    {
        int parentIndex = (i + 1) / 2 - 1;

        if (leafNodes[parentIndex].content.error > leafNodes[i].content.error)
        {
            (leafNodes[parentIndex], leafNodes[i]) = (leafNodes[i], leafNodes[parentIndex]);
        }
    }
}

private void InsertLeafNode(Node node)
{
    leafNodes.Add(node);

    for (int i = leafNodes.Count; i > 1 && leafNodes[i - 1].content.error < leafNodes[i / 2 - 1].content.error; i /= 2)
    {
        (leafNodes[i / 2 - 1], leafNodes[i - 1]) = (leafNodes[i / 2 - 1], leafNodes[i - 1]);
    }
}

public class Node
{
    public ImageRegion content;
    public Node? parent = null;
    public bool compressed = false;
    public Node?[] children = [null, null, null, null];

    public static Node DeadNode = new Node()
    {
        content = new ImageRegion(new Region2Int(), double.PositiveInfinity)
    };
}

```

#### 4.2.4. Util

```

using System.Diagnostics.CodeAnalysis;
using System.Numerics;

namespace ImageCompressor.Util;

public struct Vector2Int
{
    public int x;
    public int y;
}

```

```

public Vector2Int(int x, int y)
{
    this.x = x;
    this.y = y;
}

public static Vector2Int operator +(Vector2Int a, Vector2Int b) => new Vector2Int(a.x + b.x, a.y + b.y);
public static Vector2Int operator -(Vector2Int a, Vector2Int b) => new Vector2Int(a.x - b.x, a.y - b.y);
public static Vector2Int operator *(Vector2Int v, int s) => new Vector2Int(v.x * s, v.y * s);
public static Vector2Int operator *(int s, Vector2Int v) => v * s;
public static Vector2Int operator /(Vector2Int v, int s) => new Vector2Int(v.x / s, v.y / s);
public static bool operator ==(Vector2Int a, Vector2Int b) => a.x == b.x && a.y == b.y;
public static bool operator !=(Vector2Int a, Vector2Int b) => a.x != b.x || a.y != b.y;

public override bool Equals([NotNullWhen(true)] object? obj)
{
    if (obj is not Vector2Int) return false;

    Vector2Int v = (Vector2Int) obj;

    return v.x == x && v.y == y;
}

public override int GetHashCode()
{
    return x.GetHashCode() ^ y.GetHashCode();
}

public struct Region2Int
{
    public Vector2Int start;
    public Vector2Int end;

    public Region2Int(Vector2Int start, Vector2Int end)
    {
        this.start = start;
        this.end = end;
    }

    public Region2Int(int startX, int startY, int endX, int endY)
    {
        this.start = new Vector2Int(startX, startY);
        this.end = new Vector2Int(endX, endY);
    }

    public Vector2Int size { get => end - start + new Vector2Int(1, 1); }
    public int area { get => size.x * size.y; }
    public bool isValid { get => start.x <= end.x && start.y <= end.y; }

    public static bool operator ==(Region2Int a, Region2Int b) => a.start == b.start && a.end == b.end;
}

```

```

public static bool operator !=(Region2Int a, Region2Int b) => a.start != b.start || a.end != b.end;

public override bool Equals([NotNullWhen(true)] object? obj)
{
    if (obj is not Region2Int) return false;

    Region2Int r = (Region2Int) obj;

    return r.start == start && r.end == end;
}

public override int GetHashCode()
{
    return start.GetHashCode() ^ end.GetHashCode();
}

public struct ImageRegion
{
    public Region2Int region;
    public double error;

    public ImageRegion(Region2Int region, double error = 0d)
    {
        this.region = region;
        this.error = error;
    }
}

```

## BAB V

### IMPLEMENTASI BONUS

#### 5.1. Structural Similarity Index (SSIM)

Secara umum, *Structural Similarity Index* atau bisa juga disebut SSIM adalah indeks yang mengukur kemiripan dari dua gambar secara objektif. Namun, dikarenakan program hanya dapat melihat 1 (satu) gambar, yaitu gambar yang akan dikompresi, maka program akan melakukan ‘self-comparison’ yaitu membandingkan gambar setelah kompresi dan sebelum kompresi.

Dalam pengukuran gambar, SSIM mempertimbangkan tiga faktor utama, yaitu kecerahan (*Luminosity*), kontras (*Contrast*), dan struktur (*Structure*). Dimana kecerahan didapatkan dari perbandingan rata-rata intensitas cerahnya piksel dari dua gambar, kontras didapatkan dari perbandingan intensitas warna piksel dari dua gambar, dan struktur didapatkan dari perbandingan pola piksel dari dua gambar.

Rumus (algoritma) perhitungan SSIM yang digunakan dalam program ini yaitu:

$$SSIM(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$$

Dimana X merepresentasikan gambar sebelum kompresi dan Y merepresentasikan gambar setelah kompresi. Dan  $C_n$  merepresentasikan sebuah konstanta kecil yang ditentukan dengan rumus  $C_n = (K_n L)^2$  dengan  $K_n \ll 1$  dan L adalah rentang nilai piksel (255 untuk 8-bit tiap kanal). Dalam program ini, digunakan  $K_1 = 0.01$  dan  $K_2 = 0.03$ .

Karena gambar setelah kompresi merupakan blok warna monoton, rumus untuk menghitung SSIM dapat disederhanakan sehingga menjadi:

$$SSIM(x, y) = \frac{C_2}{\sigma_{x,c}^2 + C_2}$$

Pada gambar berwarna rumus SSIM dapat digunakan pada setiap perhitungan channel RGB dan kembali disatukan menggunakan rumus berikut:

$$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$$

Dimana R, G, dan B masing-masing merepresentasikan channel warna pada gambar dan w adalah bobot tiap *channel*. Pada program ini, diberlakukan bobot sebesar 0,299 untuk merah, 0,587 untuk hijau, dan 0,114 untuk biru.

## 5.2. Target Kompresi

Algoritma yang digunakan untuk kompresi gambar menggunakan target kompresi sedikit berbeda dengan algoritma yang menggunakan *threshold* tertentu. Algoritma kompresi menggunakan target kompresi tidak diimplementasikan menggunakan rekursi, tetapi menggunakan *looping* atas daun-daun dalam *quadtree*. Karena seluruh daun-daun *quadtree* diperlukan untuk dapat melakukan *looping*, maka program membuat representasi aktual *quadtree* dalam memori sebelum melakukan kompresi. Program mengkompresi daun-daun *quadtree* mulai dari yang memiliki error terkecil dan menandakan *node* tersebut sudah terkompresi dan mengeluarkannya dari *leaf nodes*. Ketika suatu *node* dalam *quadtree* sudah terkompresi semua *children*-nya, *node* tersebut ditambahkan dalam daftar daun *quadtree* sesuai dengan urutan errornya. Proses ini dilakukan hingga ukuran gambar menjadi kurang dari target kompresi.

## 5.3. GIF

Implementasi pembuatan GIF proses kompresi dilakukan setelah kompresi setelah. Informasi pada *quadtree* yang tersisa setelah kompresi digunakan untuk membuat GIF. Setiap *frame* dalam GIF menunjukkan kedalaman tertentu pada *quadtree*. Program mulai dari *root node* sebagai *frame* pertama dalam GIF. Untuk tiap *frame*, setiap *node* yang diproses program dalam suatu *layer* digambarkan warna rata-ratanya dalam areanya. Program kemudian menuju ke kedalaman berikutnya dan membuat *frame* baru. Proses ini diulangi hingga akhirnya program menemukan sebuah kedalaman yang semua *node*-nya sudah terkompresi.

## BAB VI

### EKSPERIMENT DAN ANALISIS

#### 6.1. Pengujian Program

File *output* (keluaran) yang berupa gambar sebelum dan setelah dikompresi serta *gif* proses kompresi dapat dilihat di dalam folder test pada repository di *link* github yang tertera pada lampiran dibawah. Berikut adalah beberapa *test case* yang dicoba pada program, beserta hasil *output* (keluaran) pada CLI berupa screenshot:

| No | CLI                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | I/O                                                                                                                                                                         |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. | <pre>Image Compressor v1.0 Enter an absolute image path: C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\input\bocchi.png Choose an error calculation method. Possible values are: &gt; Max Pixel Difference &gt; Variance &gt; Mean Absolute Deviation (MAD) &gt; Entropy &gt; SSIM Entropy Enter desired threshold: 0 Enter minimum block size: 0,5 Enter minimum block size: 1 Enter desired compression rate: 0,5 Enter an absolute output path (should have the same extension as input file): C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\bocchi.png Enter an absolute output path for GIF: C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\bocchi.gif  Compressing. Please Wait...  Compression completed in 79954,4165 milliseconds Original size: 12599954 bytes Compressed size: 6299975 bytes Compression percentage: 50,00001587307382% Quadtree max depth: 13 Quadtree nodes: 6395221</pre> | <br> |

2.

```
Image Compressor v1.0
Enter an absolute image path: C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\input\cos_Orange.jpg
Choose an error calculation method.
Possible values are:
> Max Pixel Difference
> Variance
> Mean Absolute Deviation (MAD)
> Entropy
> SSIM
Variance
Enter desired threshold: 0
Enter minimum block size: 32
Enter desired compression rate: 0,4
Enter an absolute output path (should have the same extension as input file):
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\cos_Orange.jpg
Enter an absolute output path for GIF:
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\cos_Orange.gif

Compressing. Please Wait...

Compression completed in 3041,6341 milliseconds
Original size: 248689 bytes
Compressed size: 148984 bytes
Compression percentage: 46,072965982772894%
Quadtree max depth: 8
Quadtree nodes: 21845
```



3.

```
Image Compressor v1.0
Enter an absolute image path: C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\input\shinobu.jpg
Enter an absolute image path: C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\input\shinobu.jpg
Choose an error calculation method.
Possible values are:
> Max Pixel Difference
> Variance
> Mean Absolute Deviation (MAD)
> Entropy
> SSIM
Max Pixel Difference
Enter desired threshold: 0
Enter minimum block size: 32
Enter desired compression rate: 0,5
Enter an absolute output path (should have the same extension as input file):
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\shinobu.jpg
Enter an absolute output path for GIF:
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\shinobu.gif

Compressing. Please Wait...

Compression completed in 1767,5124 milliseconds
Original size: 2108743 bytes
Compressed size: 1050354 bytes
Compression percentage: 50,00883303850635%
Quadtree max depth: 8
Quadtree nodes: 21845
```



4.

```
Image Compressor v1.0
Enter an absolute image path: C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\input\landscape.jpg
Choose an error calculation method.
Possible values are:
> Max Pixel Difference
> Variance
> Mean Absolute Deviation (MAD)
> Entropy
> SSIM
Mean Absolute Deviation (MAD)
Enter desired threshold: 0
Enter minimum block size: 1
Enter desired compression rate: 0,3
Enter an absolute output path (should have the same extension as input file):
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\landscape.jpg
Enter an absolute output path for GIF:
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\landscape.gif

Compressing. Please Wait...

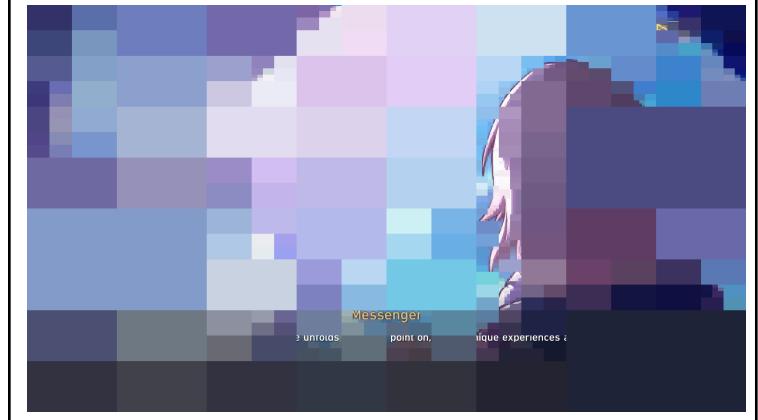
Compression completed in 75876,4219 milliseconds
Original size: 17407707 bytes
Compressed size: 12185286 bytes
Compression percentage: 30,000625584977968%
Quadtree max depth: 13
Quadtree nodes: 16905557
```





5.

```
Image Compressor v1.0
Enter an absolute image path: C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\input\march.png
Choose an error calculation method.
Possible values are:
> Max Pixel Difference
> Variance
> Mean Absolute Deviation (MAD)
> Entropy
> SSIM
Max Absolute Deviation (MAD)
Choose an error calculation method.
Possible values are:
> Max Pixel Difference
> Variance
> Mean Absolute Deviation (MAD)
> Entropy
> SSIM
Mean Absolute Deviation (MAD)
Enter desired threshold: 0.2
Enter minimum block size: 1
Enter desired compression rate: 0
Enter an absolute output path (should have the same extension as input file):
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\march.png
Enter an absolute output path for GIF:
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\march.gif
Compressing. Please Wait...
Compression completed in 4413,8203 milliseconds
Original size: 2041819 bytes
Compressed size: 49568 bytes
Compression percentage: 97,57236072345296%
Quadtree max depth: 11
Quadtree nodes: 9645
```



```

Image Compressor v1.0
Enter an absolute image path: C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\input\Cos_Pink.jpeg
Choose an error calculation method.
Possible values are:
> Max Pixel Difference
> Variance
> Mean Absolute Deviation (MAD)
> Entropy
> SSIM
Variance
Enter desired threshold: 0,01
Enter minimum block size: 1
Enter desired compression rate: 0
Enter an absolute output path (should have the same extension as input file):
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\Cos_Pink.jpeg
Enter an absolute output path for GIF:
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\Cos_Pink.gif

Compressing. Please Wait...

Compression completed in 6882,8426 milliseconds
Original size: 266281 bytes
Compressed size: 222655 bytes
Compression percentage: 16,38344456690108%
Quadtree max depth: 11
Quadtree nodes: 229213

```



6.

```

Image Compressor v1.0
Enter an absolute image path: C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\input\Furina.jpg
Choose an error calculation method.
Possible values are:
> Max Pixel Difference
> Variance
> Mean Absolute Deviation (MAD)
> Entropy
> SSIM
SSIM
Enter desired threshold: 0,5
Enter minimum block size: 1
Enter desired compression rate: 0
Enter an absolute output path (should have the same extension as input file):
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\Furina.jpg
Enter an absolute output path for GIF:
C:\Users\arlow\Documents\Stima\Tucil2_13523123_13523161\test\output\Furina.gif

Compressing. Please Wait...

Compression completed in 2457,3737 milliseconds
Original size: 825027 bytes
Compressed size: 663666 bytes
Compression percentage: 19,558269002105387%
Quadtree max depth: 11
Quadtree nodes: 156905

```

7.



## 6.2. Analisis Kompleksitas Algoritma Divide and Conquer *Quadtree*

Analisis kompleksitas algoritma yang digunakan dibagi menjadi beberapa bagian. Hal ini dilakukan karena target kompresi (*threshold* dan *percentage*) yang dipilih memengaruhi algoritma yang digunakan. Pembagian analisis kompleksitas juga mempermudah dan membuat lebih jelas perhitungan kompleksitas algoritma yang digunakan. Berikut adalah analisis kompleksitas per bagian dari algoritma program yang dibuat.

### 6.2.1. Analisis Kompleksitas Pembuatan *Quadtree*

Pembuatan *Quadtree* adalah proses yang dilakukan baik dengan target *threshold* maupun *percentage*. Operasi yang akan dihitung untuk menentukan kompleksitas algoritma

pembuatan *quadtree* adalah jumlah *node* yang diakses. *Quadtree* dibuat dengan membagi terus *root node* yaitu daerah seluruh gambar menjadi 4 *child node* berukuran sama yang dibagi lagi terus menerus hingga tidak ada lagi *node* yang dapat dibagi tanpa melanggar *minimum block size*. Jika pembuatan *quadtree* terjadi dengan sempurna (ukuran pembagian sama rata dan semua *leaf nodes* berada pada *depth* yang sama), proses ini akan membuat *leaf nodes* sebanyak:

$$\text{Number of Leaves} = \frac{\text{Image Area}}{\text{Minimum Block Size}}$$

Karena diketahui bahwa setiap 4 *leaf nodes* berasal dari 1 *parent node*, maka dapat diketahui jumlah *nodes* yang ada pada *depth* 1 tingkat lebih tinggi sebagai jumlah *leaf nodes* dibagi 4. Proses yang sama dapat dilakukan untuk mencari jumlah *nodes* pada *depth*  $i$  jika diketahui jumlah *nodes* pada *depth*  $i + 1$ . Maka, dengan  $N$  sebagai *number of leaves*, dapat ditentukan total seluruh *nodes* yang berada dalam *quadtree* sebagai:

$$\text{Nodes} = N + \frac{N}{4} + \frac{N}{16} + \dots + 1$$

$$\text{Nodes} = N\left(\frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \dots + \frac{1}{N}\right)$$

$$\text{Nodes} = N\left(\frac{1+4+16+32+\dots+N}{N}\right)$$

$$\text{Nodes} = 1 + 4 + 16 + 32 + \dots + N$$

$$\text{Nodes} = \frac{4^{\log_4(N)+1}-1}{4-1}$$

$$\text{Nodes} = \frac{1}{3}(4N - 1)$$

Sehingga didapatkan bahwa jumlah *nodes* yang perlu diakses dalam pembuatan *quadtree* adalah  $\frac{1}{3}(4N - 1)$  dengan  $N$  adalah luas gambar dibagi luas minimum. Jika  $n$  dianggap jumlah pixel dalam gambar dan  $m$  dianggap luas minimum blok, dapat ditentukan fungsi waktu dan kompleksitas waktu sebagai:

$$T(n) = \frac{4}{3m}(n - 1) \Rightarrow O(n)$$

### 6.2.2. Analisis Kompleksitas Algoritma Kompresi Berbasis *Threshold*

Algoritma kompresi berbasis *threshold* menggunakan rekursi untuk menelusuri *quadtree* yang telah dibuat mulai dari *root node* hingga semua *node* terkompresi sesuai *threshold*. Ketika suatu *node* telah dikompresi, semua *node* yang merupakan keturunan dari *node* tersebut juga dianggap telah dikompresi sehingga tidak perlu dicek lagi. Pada setiap

node yang belum terkompresi, algoritma menentukan *error* pada *node* tersebut dan menormalisasikan daerah gambar yang direpresentasikan *node* tersebut. Sama seperti pada *quadtree*, operasi yang akan dihitung dalam analisis kompleksitas ini adalah jumlah akses ke *node* tetapi ditambah pula dengan jumlah akses ke *pixel*.

Kompleksitas algoritma ini berbeda untuk *worst case scenario* dan juga *best case scenario*. Kasus *best case scenario* didapatkan ketika *node* pertama yaitu *root node* memiliki *error* yang melebihi *threshold*. Hal tersebut menyebabkan semua *node* untuk terkompresi sehingga algoritma hanya perlu mengakses 1 *node*. Pada akses 1 *node* tersebut, algoritma melakukan pengecekan *error* dan juga normalisasi daerah. Kedua operasi tersebut melakukan akses ke setiap piksel dalam implementasinya. karena itu, jika dianggap n sebagai jumlah piksel, didapatkan *best case scenario* dari algoritma ini sebagai:

$$T(n) = n + n + 1 = 2n + 1 \Rightarrow O(n)$$

Kompleksitas *worst case scenario* dari algoritma ini lebih sulit dihitung dibandingkan *best case scenario*. Pada *worst case scenario*, algoritma menelusuri semua node dalam *quadtree*. Karena semua *node* ditelusuri, dapat dianggap bahwa *node* yang terkompresi hanya *leaf nodes*. Sementara itu, semua *nodes* diakses dan juga dihitung *error*-nya oleh algoritma. Jika diasumsikan bahwa struktur *quadtree* sempurna dan pada suatu *depth* jumlah piksel yang ditutupi oleh semua *nodes* sama dengan jumlah piksel dalam gambar, dapat ditentukan bahwa jumlah akses piksel pada seluruh gambar adalah jumlah piksel dikali dengan kedalaman *quadtree*. Kedalaman *quadtree* dapat dinyatakan sebagai berapa kali *leaves* dibagi 4 hingga menjadi 1 atau  $\log_4(n)$ . Jumlah akses node juga dapat ditentukan menggunakan rumus pada bagian pembuatan *quadtree* yaitu  $(4/3m)(n - 1)$ . Jika n dianggap sebagai jumlah piksel dan m sebagai luas blok terkecil, didapatkan kompleksitas dari *worst case scenario* sebagai:

$$T(n) = \frac{4}{3m} (n - 1) + n \log_4 \left( \frac{n}{m} \right) + \frac{n}{m} + n \Rightarrow O(n \log n)$$

### 6.2.3. Analisis Kompleksitas Algoritma Kompresi Berbasis *Percentage*

Algoritma kompresi berdasarkan *percentage* terus melakukan kompresi hingga mencapai ukuran *file* yang diinginkan. Berbeda dengan kompresi berbasis *threshold*, algoritma ini memulai kompresi dari *leaf nodes*, bukan *root node*. Algoritma terus mengompresi *leaf nodes* dengan *error* terkecil hingga ukuran *file* mencapai target. Perhitungan kompleksitas algoritma ini akan mengikuti acuan yang sama dengan perhitungan kompleksitas algoritma menggunakan *threshold*.

Algoritma kompresi ini memiliki *best case scenario* yang lebih baik daripada algoritma menggunakan *threshold*. Apabila ukuran *file* yang diminta sudah sama dengan ukuran file sekarang, maka algoritma hanya akan melakukan 1 kali kompresi. Kompresi akan dilakukan pada *leaf node* dengan *error* terkecil yang seringkali menutupi jumlah piksel kurang lebih sama dengan luas blok minimum. Akibatnya, *best case scenario* untuk algoritma ini dapat dinyatakan sebagai:

$$T(n) = 1 + m \Rightarrow O(1)$$

Berbeda dengan algoritma berbasis *threshold*, terdapat beberapa perhitungan lain yang diperlukan untuk menentukan kompleksitas untuk *worst case scenario* algoritma ini. Perhitungan-perhitungan yang sama dari algoritma berbasis *threshold* tetap digunakan karena *worst case scenario* pada algoritma ini juga menelusuri keseluruhan *quadtree*. Namun, terdapat kompleksitas tambahan yang terjadi karena algoritma perlu memilih *leaf node* dengan *error* terkecil. *Leaf nodes* dalam *quadtree* disimpan dalam suatu *binary heap*. *Binary heap* ini memungkinkan pencarian *leaf node* dengan *error* terkecil dan penambahan *leaf node* dengan akses *node* sebanyak *leave nodes* yang ada pada saat itu. Setiap kali algoritma mengkompresi *leaf node*, *node* itu dikeluarkan dari *leaf nodes* dan *parent*-nya ditambahkan jika tidak semua *children*-nya sudah dikompres. Karena itu, kompleksitas untuk *worst case scenario* menjadi:

$$T(n) = \frac{16+3m}{12m} (n - 1) \log_4 \left(\frac{n}{m}\right) + n \log_4 \left(\frac{n}{m}\right) + \frac{n}{m} + n \Rightarrow O(n \log n)$$

Perlu diketahui bahwa pada algoritma ini, terdapat banyak operasi *save* dan *load* karena perlu mengukur ukuran *file*. Hal ini menyebabkan algoritma berjalan lebih pelan tetapi tidak dimasukkan dalam analisis kompleksitas karena implementasi dan kompleksitas waktu dari operasi tersebut tidak diketahui. Karena itu, waktu yang dipakai untuk *save* dan *load* tidak diperhitungkan dalam analisis kompleksitas.

### 6.2.3. Analisis Kompleksitas Pembuatan GIF

Pembuatan GIF adalah proses terakhir yang dilakukan oleh *Image Compressor*. Proses ini menggunakan struktur *quadtree* yang sudah dibuat sebelumnya dan menyimpan informasi mengenai *node* yang terkompresi. Proses ini memiliki kompleksitas *best case* dan *worst case scenario* yang sama dengan algoritma kompresi berbasis *threshold*. Hal ini dikarenakan proses pembuatan GIF dimulai dari *root node* sama seperti dengan algoritma kompresi *threshold* dan dapat menelusuri seluruh *quadtree* dalam keadaan terburuk. Namun, beberapa perhitungan dihilangkan untuk pembuatan GIF karena perhitungan *error* sudah tidak

perlu dilakukan lagi. Dengan demikian, kompleksitas *best case scenario* pembuatan GIF dapat dinyatakan sebagai berikut:

$$T(n) = n + 1 = n + 1 \Rightarrow O(n)$$

Sementara kompleksitas *worst case scenario* untuk pembuatan GIF dapat dinyatakan sebagai berikut:

$$T(n) = \frac{4}{3m} (n - 1) + n \log_4 \left(\frac{n}{m}\right) \Rightarrow O(n \log n)$$

#### 6.2.3. Analisis Kompleksitas Keseluruhan

Untuk mendapatkan waktu keseluruhan algoritma, dapat menjumlahkan waktu untuk pembuatan *quadtree*, kompresi *threshold/percentage*, dan pembuatan GIF. Namun, lebih sederhana apabila mendapatkan kompleksitas waktu dibandingkan dengan waktu pada algoritma. Kompleksitas waktu dapat diperoleh dengan memilih kompleksitas waktu yang terbesar dari semua bagian. Untuk algoritma ini, pilihan menggunakan kompresi berbasis *threshold* maupun *percentage* memberikan hasil yang sama. Kompleksitas waktu untuk keseluruhan algoritma adalah:

**Best Case:**  $O(n)$

**Worst Case:**  $O(n \log n)$

## LAMPIRAN

Checklist Spesifikasi Program

Tabel 7.1. Tabel *checklist* spesifikasi program

| No | Poin                                                                                   | Ya                                  | Tidak                    |
|----|----------------------------------------------------------------------------------------|-------------------------------------|--------------------------|
| 1  | Program berhasil dikompilasi tanpa kesalahan                                           | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 2  | Program berhasil dijalankan                                                            | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 3  | Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan            | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 4  | Mengimplementasi seluruh metode perhitungan error                                      | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 5  | Implementasi persentase kompresi sebagai parameter tambahan                            | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 6  | Implementasi <i>Structural Similarity Index</i> (SSIM) sebagai metode pengukuran error | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 7  | Output berupa GIF Visualisasi Proses pembentukan <i>Quadtree</i> dalam Kompresi Gambar | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 8  | Program dan laporan dibuat (kelompok) sendiri                                          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |

Pembagian Tugas

Tabel 7.2. Tabel pembagian tugas

| Tugas              | NIM                |
|--------------------|--------------------|
| Github             | 13523123, 13523161 |
| Image Compression  | 13523123, 13523161 |
| Error Calculation  | 13523123. 13523161 |
| [BONUS] SSIM       | 13523123           |
| [BONUS] GIF        | 13523161           |
| [BONUS] Persentase | 13523161           |
| Laporan            | 13523123, 13523161 |

Repository

Github: [Repository](#)

(raw link: [https://github.com/Arlow5761/Tucil2\\_13523123\\_13523161](https://github.com/Arlow5761/Tucil2_13523123_13523161))

### 7 Maret:



## **DAFTAR PUSTAKA**

Munir, Rinaldi. 2025. Algoritma Divide and Conquer (Bagian 1). [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). Diakses pada 11 April 2025.

Munir, Rinaldi. 2025. Algoritma Divide and Conquer (Bagian 2). [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf). Diakses pada 11 April 2025.

Munir, Rinaldi. 2025. Algoritma Divide and Conquer (Bagian 3). [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf). Diakses pada 11 April 2025.

Munir, Rinaldi. 2025. Algoritma Divide and Conquer (Bagian 4). [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf). Diakses pada 11 April 2025.