

# LAPORAN TUGAS KECIL

## IF2211 Strategi Algoritma


### Kompresi Gambar dengan Quadtree

Dipersiapkan oleh:

- Brian Albar Hadian                      13523048

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

|   |   |                       |   |                  |
|---|---|-----------------------|---|------------------|
|  | Sekolah Teknik Elektro<br>dan Informatika ITB | Nomor Dokumen         |   | Halaman          |
|   |   | IF2211-TK-13523048-01 |   | 39               |
|   |   | Revisi                | 0 | 24 Februari 2025 |

# 1 Ringkasan

Pembuatan program berbasis CLI (command-line interface) dalam bahasa C# ini berfungsi untuk melakukan kompresi terhadap masukan gambar menggunakan algoritma *divide and conquer*, struktur data QuadTree, dan Library ImageSharp yang dapat dijalankan dalam environment seperti UNIX dan Windows dengan menggunakan DOTNET Environment.

Dalam laporan ini, dapat ditemui beberapa hal yang saya implementasikan dalam program berbasis bahasa C#. Diantaranya:

- Spesifikasi fitur utama,
- Struktur Objek,
- Pengetesan, dan
- Pengerjaan

## 2 Penjelasan Tambahan Spesifikasi Tugas

### 2.1 *Kompresi gambar menggunakan Quadtree*

Fitur kompresi gambar diterapkan dengan menggunakan konsep Quadtree dengan nama kelas ImageTree dan menerapkan algoritma Divide and Conquer dengan bahasa C#. Program akan menerima masukan berupa gambar dengan format png, jpeg, tiff, tga, qoi, dan bmp. Kompresi kemudian dilakukan dengan cara menginisialisasi suatu ImageTree yang memiliki atribut posisi x dan y, warna rgba, ChildTree, dan parent. Struktur data ChildTree berisi atribut LeftBottom, RightBottom, LeftTop, dan RightTop dengan masing-masing bertipe data ImageTree. Selanjutnya, pixel setiap posisi dari masukan gambar akan dibaca dan dilakukan pemetaan pada ImageTree yang menyimpan posisi dan kode warna pada koordinat tersebut. Kemudian, apabila komposisi rgba pada ImageTree terkini masih melebihi suatu threshold, maka ImageTree tersebut akan membaca kembali gambar dengan membagi gambar menjadi empat bagian utama dengan ukuran lebar dan tinggi dibagi dua, masing-masing kemudian diproses kembali dengan cara yang sama hingga diperoleh ukuran terkecil yang diberikan oleh pengguna melalui masukan atau diperoleh nilai *error detection* lebih kecil dari *threshold*. Pembagian gambar menjadi empat bagian utama merupakan penerapan dari algoritma Divide and Conquer dalam membentuk gambar yang terkompresi. Selanjutnya, menggunakan ImageTree yang sudah selesai diproses, akan dibangun kembali suatu gambar dengan dimensi yang sama dengan mengikuti kode warna dari struktur ImageTree tersebut. Keluaran dari proses ini akan memberikan gambaran akhir yang dibentuk dari ImageTree yang telah diperoleh sebelumnya.

Metode error detection yang digunakan diantaranya adalah *Variance*, *Mean Absolute Deviation*, *Entropy*, dan *Max Pixel Difference*.

### 2.2 *Pemrosesan gambar terkompresi dalam bentuk GIF*

Fitur pemrosesan gambar terkompresi dalam bentuk GIF merupakan fitur tambahan yang dapat mengembalikan hasil pembentukan gambar terkompresi dalam format GIF. GIF tersebut akan berisi proses setiap tahap pembentukan frame menggunakan ImageTree yang telah dimiliki pada akhir pemrosesan. Keluaran kemudian akan disimpan pada directory yang diberikan dengan format nama sesuai dengan kehendak pengguna.

### 3 Strategi Penyelesaian

Program yang dibuat menerima berbagai bentuk data dan menerapkannya dalam bentuk objek dengan berfokus pada pengembangan berorientasi objek. Objek yang penulis gunakan diantaranya:

#### 3.1 *ImageTree*

Objek *ImageTree* dibangun atas bentuk representasi *QuadTree* yang berguna untuk menyimpan informasi dari pixel yang telah diproses dan menyimpan struktur *ChildTree* yang menyimpan *ImageTree* selanjutnya yang akan diproses. Objek ini dibentuk dengan atribut posisi *x* dan *y*, *rgba* bertipe data *<Rgba32>*, *parent* bertipe data *ImageTree*, dan *child* bertipe data *ChildTree*. Objek ini digunakan untuk memanipulasi data yang berkaitan dengan gambar, yakni pixel, kemudian melakukan operasi terhadap pixel tersebut berdasarkan metode *error detection* yang dipilih atau batas ukuran minimum gambar yang diproses. Spesifikasi tersebut diterapkan dengan berbagai fitur, diantaranya :

1. *getTotalNode(ImageTree, int)* : memperoleh total simpul pada *ImageTree* terbesar
2. *getImageTreeDepth(ImageTree root)* : memperoleh kedalaman pohon terbesar
3. *BuildTree(ref ImageTree root, int minWidth, int minHeight, double threshold, int? errorDetectionMethod, ref ImageTree top)* : membentuk *ImageTree* berdasarkan metode *error detection* dan batas ukuran minimum gambar yang dipilih.
4. *BuildImageFromImageTree(ImageTree? root, ref Image<Rgba32> constructImage)* : membentuk gambar baru terkompresi berdasarkan *ImageTree* yang telah diproses
5. *BuildImageGIFFromImageTree(ImageTree? root, bool? GIFRequest, ref Image<Rgba32> GIFConstructImage, ref int currGIFFrame)* : membentuk GIF yang berisi frame mengenai proses pembentukan gambar terkompresi.
6. *SSIMImageTree(ImageTree root, ref ImageTree top)* : mengembalikan nilai *error detection* berdasarkan *ImageTree* yang telah dimiliki dengan membandingkan pada gambar asli menggunakan metode SSIM
7. *ImageTreeVariance(ImageTree root)* : mengembalikan nilai *error detection* terhadap *ImageTree* terkini menggunakan perbandingan variansi pada pixel pada *ImageTree* terkini.
8. *VarianceImageTreeColorChannel(ImageTree root, char colorChannel)* : mengembalikan nilai variansi suatu *ImageTree* terkini berdasarkan kanal warna yang diberikan.
9. *SSIMCovarianceColorChannelImageTree(ImageTree root, ref ImageTree top, char colorChannel)* : mengembalikan kovariansi yang diperoleh berdasarkan kanal warna yang diberikan yang digunakan dalam metode
10. *MeanChannelColorValue(ImageTree root, char colorChannel)* : mengembalikan rerata dari suatu *ImageTree* pada ukuran yang dimiliki oleh masukan bertipe *ImageTree* pada kanal warna tertentu.
11. *ImageTreeMeanAbsoluteDeviation(ImageTree root)* : mengembalikan hasil dari metode *error detection Mean Absolute Deviation (MAD)* dari masukan *ImageTree*.

12. `MeanAbsoluteDeviationImageTreeColorChannel(ImageTree root, char colorChannel)` : mengembalikan hasil dari metode *error detection Mean Absolute Deviation* (MAD) dari masukan ImageTree pada kanal warna tertentu.
13. `MaxPixelDifferenceImageTree(ImageTree root)` : mengembalikan hasil dari metode *error detection Max Pixel Difference* (MPD) dari masukan ImageTree.
14. `MaxValueImageTreeColorChannel(ImageTree root, char colorChannel)` : mengembalikan nilai rgba tertinggi pada suatu masukan ImageTree pada kanal warna tertentu.
15. `MinValueImageTreeColorChannel(ImageTree root, char colorChannel)` : mengembalikan nilai rgba terendah pada suatu masukan ImageTree pada kanal warna tertentu.
16. `EntropyImageTree(ImageTree root)` : mengembalikan hasil dari metode *error detection Entropy* dari masukan ImageTree.
17. `EntropyImageTreeColorChannel (ImageTree root, char colorChannel)` : mengembalikan hasil dari metode *error detection Entropy* dari masukan ImageTree pada kanal warna tertentu.
18. `NormalizeImageTree(ImageTree root)` : melakukan normalisasi pada ImageTree yang sudah mencapai batas ukuran gambar minimum ataupun yang sudah memenuhi threshold.
19. `printImageTree(ImageTree root)` : melakukan penulisan atribut dari masukan ImageRoot pada layar.

Lokasi : `src/ImageTree.cs` (.cs & .csproj)

### 3.2 *ChildTree*

Objek *ChildTree* dibangun untuk merepresentasikan hasil pembagian oleh algoritma *divide and conquer* yang diisi oleh data bertipe ImageTree melalui pemrosesan *BuildTree*. Objek ini memiliki empat atribut, yakni *LeftBottom*, *RightBottom*, *LeftTop*, dan *RightTop*, masing-masing bertipe data ImageTree. Objek ini digunakan untuk menyimpan data ImageTree hasil dari penerapan algoritma *divide and conquer* yang berkaitan dengan pembagian ukuran lebar dan tinggi menjadi dua, warna piksel, dan parent dari *ChildTree*. Objek *ChildTree* hanya memiliki atribut dan konstruktor *default*.

Lokasi : `src/ImageTree.cs` (.cs & .csproj)

### 3.3 *Main*

Spesifikasi dari program utama adalah menerima masukan berupa gambar dengan format tertentu kemudian melakukan pemrosesan melalui ImageTree untuk memperoleh gambar yang terkompresi. Selain itu, program ini juga berfungsi untuk menyimpan kembali gambar yang telah direkonstruksi ulang menggunakan ImageTree yang sudah dibentuk. Program ini memiliki fungsi utama `Main(string[] args)` untuk menjalankan keseluruhan program pemrosesan gambar terkompresi.

Lokasi: `src/ImageQuadProcessing` (.cs & .csproj)

## 4 Strategi Penyelesaian

### 4.1 Konsep Dasar

Strategi penyelesaian yang digunakan adalah strategi algoritma *divide and conquer*. Strategi ini berfokus pada penyelesaian suatu masalah dengan membagi masalah tersebut menjadi ukuran yang lebih kecil hingga memenuhi suatu parameter tertentu, kemudian menyelesaikan masalah tersebut apabila masalah sudah dipartisi hingga batas ukuran tertentu. Strategi ini cocok digunakan untuk suatu permasalahan yang diselesaikan secara rekursif, yakni bagian-bagian yang lebih kecil dapat diselesaikan menggunakan cara yang sama pada bagian-bagian yang lebih besar. Selain itu, strategi ini cocok digunakan untuk masalah yang memiliki karakteristik independen terhadap partisinya sendiri sehingga tidak akan memengaruhi proses rekursi pada saat pemrosesan. Karena sifatnya yang membagi setiap masalah menjadi ukuran yang lebih kecil, algoritma ini biasa digunakan untuk masalah-masalah umum yang sering terjadi dan bersifat independen, seperti proses searching, pencarian nilai maksimum/minimum, dan banyak lagi lainnya.

Strategi algoritma *divide and conquer* biasa digunakan dalam berbagai persoalan, diantaranya *Large Integer Multiplication*, *Fast Fourier Transform*, *Closest Pair of Points*, *inversion pair*, dan *sorting*. Namun, strategi ini juga memiliki beberapa kelemahan, diantaranya adalah penggunaan sumber daya memori yang cukup besar, pencarian solusi yang waktu lama untuk kasus pemrosesan dengan skala besar, serta ketergantungan terhadap spesifikasi perangkat keras.

### 4.2 Penerapan

Pemrosesan kompresi gambar menggunakan struktur data ImageTree dilakukan dengan pendekatan algoritma *divide and conquer*.

Penyelesaian ini dapat dijabarkan sesuai dengan tahapan sebagai berikut.

1. Lakukan pembacaan terhadap masukan gambar
  - a. Hitung nilai metode *error detection* yang telah dipilih (*Variance*, *Mean Absolute Deviation*, *Max Pixel Difference*, *Entropy*, *SSIM*) untuk seluruh kanal warna pada ukuran yang telah dicakup oleh ImageTree
    - i. Apabila nilai metode *error detection* melebihi masukan threshold dari pengguna, dilakukan partisi gambar menjadi empat bagian bertipe ImageTree :
      1. LeftBottom dengan atribut koordinat pusat baru terletak pada koordinat yang sama pada *parent*, warna mengikuti rgba pada koordinat terbaru, ukuran lebar adalah ukuran lebar ImageTree masukan / 2, ukuran tinggi adalah ukuran tinggi ImageTree masukan / 2, dengan *parent* di-assign pada ImageTree masukan, dan dengan *child* di-assign dengan nilai null.
      2. RightBottom dengan atribut koordinat pusat baru terletak pada koordinat yang sama pada *parent*, warna mengikuti rgba pada koordinat terbaru, ukuran lebar adalah ukuran lebar ImageTree

- masukan – ukuran lebar LeftBottom, ukuran tinggi adalah ukuran tinggi ImageTree masukan / 2, dengan *parent* di-assign pada ImageTree masukan, dan dengan *child* di-assign dengan nilai null.
3. LeftTop dengan atribut koordinat pusat baru terletak pada koordinat yang sama pada *parent*, warna mengikuti rgba pada koordinat terbaru, ukuran lebar adalah ukuran lebar ImageTree masukan / 2, ukuran tinggi adalah ukuran tinggi ImageTree masukan – ukuran tinggi LeftBottom, dengan *parent* di-assign pada ImageTree masukan, dan dengan *child* di-assign dengan nilai null.
  4. LeftBottom dengan atribut koordinat pusat baru terletak pada koordinat yang sama pada *parent*, warna mengikuti rgba pada koordinat terbaru, ukuran lebar adalah ukuran lebar ImageTree masukan – ukuran lebar LeftBottom, ukuran tinggi adalah ukuran tinggi ImageTree masukan – ukuran tinggi LeftBottom, dengan *parent* di-assign pada ImageTree masukan, dan dengan *child* di-assign dengan nilai null.
    - a. Selanjutnya, dilakukan perhitungan terhadap metode *error detection* yang sama yang telah diberikan sebelumnya dan tahap kembali ke bagian (a) teratas.
    - ii. Apabila nilai metode *error detection* melebihi masukan threshold dari pengguna, dilakukan normalisasi terhadap pixel gambar yang diisi dengan rata-rata dari rgba pada ukuran gambar yang dicakup oleh ImageTree dan atribut *child* di-assign dengan null.
    - b. Apabila nilai metode *error detection* melebihi masukan threshold dari pengguna, dilakukan normalisasi terhadap pixel gambar yang diisi dengan rata-rata dari rgba pada ukuran gambar yang dicakup oleh ImageTree dan atribut *child* di-assign dengan null.
  2. Setelah ImageTree selesai diproses, kemudian dilakukan pembentukan ulang gambar menggunakan hasil ImageTree dengan algoritma sebagai berikut.
    - a. Dibentuk suatu gambar baru NewImage dengan dimensi yang sama seperti sebelumnya.
    - b. Lakukan pembacaan terhadap ImageTree
      - i. Jika atribut *child* dari masukan ImageTree tidak null, maka dilakukan pembacaan kembali pada atribut *child* dan kembali ke tahap b.
      - ii. Jika atribut *child* dari masukan ImageTree berisi null, maka dilakukan pengisian warna pada gambar baru NewImage dimulai dari koordinat x pada masukan ImageTree hingga lebar dari ImageTree dan koordinat y pada masukan ImageTree hingga tinggi dari ImageTree.
        1. Kembali pada tahap b.

Hasil penerapan kemudian akan dikembalikan dalam bentuk keluaran dengan format gambar. Jika tidak diperoleh hasil sama sekali, maka program akan mengembalikan pesan kesalahan dan keluar dari program.

### 4.3 Analisis

Berdasarkan algoritma yang digunakan, pada tahap *divide*, setiap pemrosesan ImageTree akan menghasilkan empat struktur data ImageTree yang menjadi atribut *child*. Selanjutnya, pada tahap *solve*, ImageTree akan diproses menggunakan metode NormalizeImageTree pada objek ImageTree yang memiliki waktu konstan bergantung pada ukuran dari ImageTree dan *bukan* dari banyaknya partisi yang terjadi. Terakhir, pada tahap *combine*, ImageTree akan diproses dan digabungkan dengan melakukan *assignment* pada atribut *child* dengan data bertipe ChildTree dengan atribut sesuai partisi yang dilakukan.

Maka berdasarkan hal tersebut, dengan meninjau jumlah masukan, maka rumusan kompleksitas waktu dapat dinyatakan sebagai berikut.

$$T(n) = 4T\left(\frac{n}{4}\right) + c$$

Dengan mengikuti teorema master, diperoleh kompleksitas waktu sebesar

$$O(n^{\log_4 4}) = O(n)$$

### 4.4 Pseudocode

{ kompresi gambar menggunakan QuadTree }

**Procedure** ImageQuadProcessing (input I : ImageTree, input MinSize : integer, input threshold : double, output img : Image)

{ Spesifikasi : melakukan pemrosesan terhadap gambar masukan untuk membangun QuadTree berdasarkan error detection method tertentu }

{ Kamus }

{ ErrorDetectionMethod(input I : ImageTree) -> integer : fungsi yang mengembalikan hasil perhitungan terhadap error detection }

{ NormalizeImageTree(input I : ImageTree) : procedure yang melakukan normalisasi terhadap seluruh pixel yang tercakup oleh I }

{ Algoritma }

if (GetSize(I) > MinSize then

if (ErrorDetectionMethod(I) > threshold) then

ImageQuadProcessing(I.LeftBottom, MinSize, threshold, img);

ImageQuadProcessing(I.RightBottom, MinSize, threshold, img);

ImageQuadProcessing(I.LeftTop, MinSize, threshold, img);

ImageQuadProcessing(I.RightTop, MinSize, threshold, img);

else

NormalizeImageTree(I);

else

NormalizeImageTree(I);

img <- constructImageFromImageTree(I);

}



## 5 Data Test

### 5.1 Test 1

- File: test.jpg
- Sebelum kompresi :



- Setelah kompresi :



- Output

Enter your photo path :

H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\input\test.jpeg

Enter your preferred method of error detection :

1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy

2

Enter output path :

```
H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\output\test_output.jpeg
```

```
Do you want to make a GIF too?(Y/N)
```

```
N
```

```
Enter minimum size block :
```

```
3
```

```
Enter minimum threshold :
```

```
6
```

```
Time elapsed : 78 ms
```

```
Total node : 273
```

```
Tree depth : 6
```

```
File size before compression : 10,87890625 kb
```

```
File size before compression : 11,671875 kb
```

```
Compression rate : -0,07289048473967674
```

```
Your image processed successfully and saved at  
../test/output/test_output
```

## 5.2 Test 2

- File : test2.txt



- Output

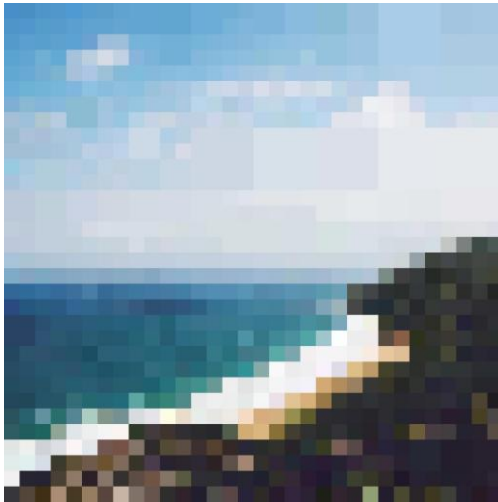
```
Enter your photo path :
```

```
H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\input\test2.jpg
```

```
Enter your preferred method of error detection :
```

1. Variance
2. Mean Absolute Deviation

```
3. Max Pixel Difference
4. Entropy
2
Enter output path :
H:\Visual Studio Code Data\C#\Academia\tucil2-
13523048\test\output\test2_output.jpg
Do you want to make a GIF too?(Y/N)
N
Enter minimum size block :
10
Enter minimum threshold :
3
Time elapsed : 404 ms
Total node : 266
Tree depth : 6
File size before compression : 82,052734375 kb
File size before compression : 18,365234375 kb
Compression rate : 0,7761776677536836
Your image processed successfully and saved at
../test/output/test2_output
```



### 5.3 Test 3

- File : test3.jpg



- **Output**

Enter your photo path :

H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\input\test3.jpg

Enter your preferred method of error detection :

1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM

1

Enter output path :

H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\output\test3\_output.jpg

Do you want to make a GIF too?(Y/N)

N

Enter minimum size block :

4

Enter minimum threshold :

2

Time elapsed : 1045 ms

Total node : 5461

Tree depth : 8

File size before compression : 148,0654296875 kb

File size before compression : 30,8486328125 kb

Compression rate : 0,7916553993892586

Your image processed successfully and saved at H:\Visual  
Studio Code Data\C#\Academia\tucil2-  
13523048\test\output\test3\_output.jpg



## 5.4 Test 4

- File : test4.jpeg



- Output

Enter your preferred method of error detection :

1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy

4

Enter output path :

H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\output\test4\_output.jpg

Do you want to make a GIF too?(Y/N)

N

Enter minimum size block :

5

Enter minimum threshold :

2

Time elapsed : 350 ms

Total node : 1364

Tree depth : 7

File size before compression : 242,896484375 kb

File size before compression : 23,072265625 kb

Compression rate : 0,9050119408505746

Your image processed successfully and saved at

../test/output/test4\_output





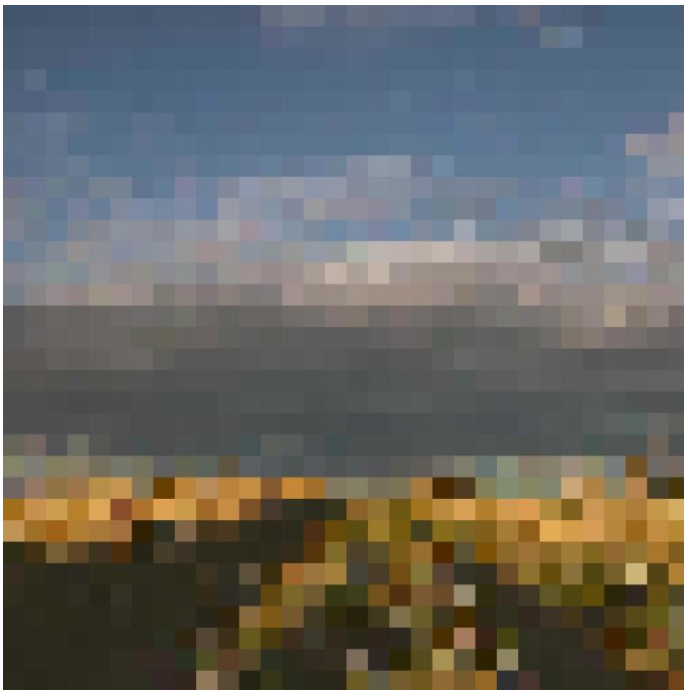
### 5.5 Test 5

- File : test5.jpg



- Output  
Enter your photo path :  
H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\input\test5.jpg  
Enter your preferred method of error detection :  
1. Variance

```
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
1
Enter output path :
H:\Visual Studio Code Data\C#\Academia\tucil2-
13523048\test\output\test5_output.jpg
Do you want to make a GIF too?(Y/N)
N
Enter minimum size block :
9
Enter minimum threshold :
1
Time elapsed : 776 ms
Total node : 341
Tree depth : 6
File size before compression : 88,931640625 kb
File size before compression : 18,4365234375 kb
Compression rate : 0,7926888190982364
Your image processed successfully and saved at
../test/output/test5_output
```



## 5.6 Test 6

- File : test6.jpg





- Output

Enter your photo path :

H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\input\test6.jpg

Enter your preferred method of error detection :

1. Variance

2. Mean Absolute Deviation

3. Max Pixel Difference

4. Entropy

3

Enter output path :

H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\output\test6\_output.jpg

Do you want to make a GIF too?(Y/N)

N

Enter minimum size block :

6

Enter minimum threshold :

4

Time elapsed : 307 ms

Total node : 1365

Tree depth : 7

File size before compression : 143,1357421875 kb

File size before compression : 20,8330078125 kb

Compression rate : 0,8544527907976339

Your image processed successfully and saved at ../test/output/test6\_output



### 5.7 Test 7

- File : profile.jpg



- Output

Enter your photo path :

H:\Visual Studio Code  
Data\C#\Academia\tucil2-  
13523048\test\input\profile.jpg

Enter your preferred method of  
error detection :

1. Variance
2. Mean Absolute Deviation

3. Max Pixel Difference

4. Entropy

5. SSIM

1

Enter output path :

H:\Visual Studio Code  
Data\C#\Academia\tucil2-  
13523048\test\output\profile\_outp  
ut.jpg

Do you want to make a GIF  
too? (Y/N)

Y

Enter minimum size block :

10

Enter minimum threshold :

5

Enter your output path for GIF :

H:\Visual Studio Code  
Data\C#\Academia\tucil2-  
13523048\test\output\profile\_outp  
ut\_gif.gif

Time elapsed : 37120 ms

Total node : 341

Tree depth : 6

File size before compression :  
91,962890625 kb

File size before compression :  
40,203125 kb

Compression rate :  
0,5628331740469363

Your image processed successfully  
and saved at H:\Visual Studio  
Code Data\C#\Academia\tucil2-  
13523048\test\output\profile\_outp  
ut.jpg

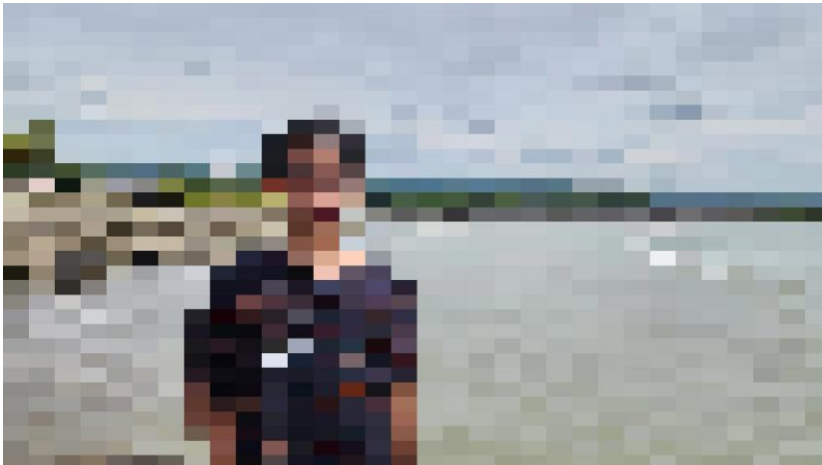
Your GIF image processed  
successfully and saved at  
H:\Visual Studio Code

Data\C#\Academia\tucil2-  
13523048\test\output\profile\_outp  
ut\_gif.gif

JPG



GIF (May be seen in the folder)



## 5.8 Test 8

- File : test7.jpg



- **Output**

Enter your photo path :

H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\input\test7.jpg

Enter your preferred method of error detection :

1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM

5

Enter output path :

H:\Visual Studio Code Data\C#\Academia\tucil2-13523048\test\output\test7\_output.jpg

Do you want to make a GIF too? (Y/N)

N

Enter minimum size block :

5

Enter minimum threshold :

0.2

Res :

0,0022912916492896530,0049171577313106510,00095121  
18142219

Time elapsed : 555 ms

Total node : 0

Tree depth : 1

File size before compression : 77,375 kb

File size before compression : 16,59765625 kb

Compression rate : 0,7854907108239095

Your image processed successfully and saved at  
H:\Visual Studio Code Data\C#\Academia\tucil2-  
13523048\test\output\test7\_output.jpg



- Penjelasan

- Pada iterasi pertama, langsung dicapai nilai threshold sehingga hanya terbentuk satu warna untuk keseluruhan pixel pada dimensi tersebut.

## 6 Lampiran

### 6.1 Repository Sumber Kode

Github : [BrianHadianSTEI23/tucil2-13523048](https://github.com/BrianHadianSTEI23/tucil2-13523048):

### 6.2 Tabel Verifikasi

| No | Poin  | Ya | Tidak |
|----|---|----|-------|
| 1  | Program berhasil dikompilasi tanpa kesalahan  | v  |       |
| 2  | Program berhasil dijalankan   | v  |       |
| 3  | Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan             | v  |       |
| 4  | Mengimplementasi seluruh metode perhitungan error wajib                                 | v  |       |
| 5  | [Bonus] Implementasi persentase kompresi sebagai parameter tambahan                     |    | v     |
| 6  | [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error | v  |       |
| 7  | [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar | v  |       |
| 8  | Program dan laporan dibuat (kelompok) sendiri   | v  |       |