Implementasi Algoritma Huffman untuk Optimasi Kompresi Data pada Penyimpanan Citra Digital

Carlo Angkisan - 13523091[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523091@std.stei.itb.ac.id, [2]carloangkisan21@gmail.com

Abstrak—Di era digital, kebutuhan akan penyimpanan data yang efisien semakin meningkat, terutama untuk citra digital yang memiliki ukuran file besar. Makalah ini membahas penerapan Algoritma Huffman sebagai salah satu metode *lossless compression* untuk mengoptimalkan proses kompresi data pada citra digital *grayscale*. Proses implementasi meliputi konversi citra ke dalam matriks piksel, perhitungan frekuensi intensitas piksel, pembentukan pohon Huffman, serta pengkodean dan dekode data. Dengan memanfaatkan distribusi frekuensi intensitas piksel, Algoritma Huffman mampu menghasilkan pola pengkodean yang lebih efisien. Hasil pengujian menunjukkan bahwa algoritma ini efektif untuk mengurangi redundansi data tanpa kehilangan informasi pada citra digital. Untuk pengembangan lebih lanjut, kombinasi Algoritma Huffman dengan algoritma lain, seperti *Run-Length Encoding* (RLE), dapat menjadi solusi untuk meningkatkan efisiensi dan performa kompresi secara keseluruhan.

Kata Kunci—Algoritma Huffman, Kompresi Data, Citra *Grayscale*, Penyimpanan.

I. PENDAHULUAN

Di era digital saat ini, jumlah file digital, khususnya foto atau citra terus mengalami peningkatan seiring dengan berkembangnya teknologi dan kebutuhan masyarakat. Citra digital kini menjadi bagian penting dalam berbagai aktivitas, mulai dari dokumentasi pribadi, media sosial, hingga kebutuhan profesional seperti desain grafis dan fotografi. Namun, peningkatan ini juga menimbulkan tantangan, terutama dalam hal efisiensi penyimpanan, mengingat ukuran file citra cenderung besar dan memerlukan kapasitas penyimpanan yang signifikan.

Citra digital yang direpresentasikan dalam bentuk piksel sebagai elemen terkecil, memiliki ukuran data yang bergantung pada resolusi dan kualitas gambar. Kualitas yang lebih tinggi sering kali berbanding lurus dengan ukuran file yang semakin besar. Hal ini dapat menyebabkan permasalahan dalam pengelolaan ruang penyimpanan, terutama pada perangkat dengan kapasitas terbatas atau pada server yang menyimpan data dalam jumlah besar.

Untuk mengatasi permasalahan tersebut, diperlukan metode kompresi data yang efektif. Kompresi data bertujuan untuk mengurangi ukuran file tanpa menghilangkan informasi penting yang terkandung di dalamnya, sehingga file menjadi lebih hemat ruang penyimpanan dan lebih efisien untuk proses transmisi data melalui jaringan. Proses kompresi juga memungkinkan pengelolaan data dalam jumlah besar menjadi lebih praktis dan ekonomis.

Makalah ini bertujuan untuk mengeksplorasi penerapan Algoritma Huffman sebagai salah satu metode *lossless compression* yang dapat digunakan untuk mengoptimalkan penyimpanan citra digital. Algoritma Huffman bekerja dengan memanfaatkan frekuensi kemunculan simbol dalam data untuk menghasilkan pola pengkodean bit yang efisien. Dengan teknik ini, ukuran file citra dapat diperkecil secara signifikan tanpa mengorbankan informasi, sehingga memberikan solusi yang relevan terhadap kebutuhan efisiensi penyimpanan dalam pengelolaan file digital saat ini.

II. DASAR TEORI

2.1. Citra Digital

Citra adalah gambaran dua dimensi (dwimatra) yang dapat direpresentasikan secara matematis sebagai fungsi $f(x, y)$, di mana x dan y mewakili koordinat pada bidang dua dimensi, sedangkan $f(x, y)$ menunjukkan tingkat intensitas cahaya pada titik tertentu. Citra dapat berupa sinyal dwimatra kontinu yang dapat diamati langsung oleh sistem visual manusia. Dalam bentuk digital, citra dihasilkan dari sistem perekaman sinyal yang menghasilkan keluaran berupa gambar optik seperti foto, gambar analog seperti tampilan televisi, atau citra digital yang disimpan di media penyimpanan seperti *disk* atau pita magnetik.

Berdasarkan sifatnya, citra dibagi menjadi dua jenis utama, yaitu citra diam (*still image*) dan citra bergerak (*moving image*). Citra diam merujuk pada gambar tunggal yang statis, sementara citra bergerak adalah kumpulan citra diam yang ditampilkan secara berurutan untuk menciptakan ilusi gerakan. Dalam citra bergerak, setiap gambar individual dalam rangkaian disebut sebagai *frame*. Kumpulan *frame* ini sering digunakan dalam aplikasi video atau animasi.

Citra digital diperoleh melalui proses pencuplikan

(*sampling*) pada domain ruang dan waktu. Pencuplikan ruang dilakukan dengan membagi citra ke dalam koordinat (x, y), sedangkan pencuplikan ruang dan waktu menghasilkan serangkaian *frame* yang membentuk video digital. Dengan pencuplikan ini, citra digital dapat dianalisis dan diolah oleh perangkat elektronik, memungkinkan berbagai aplikasi dalam teknologi modern.

Dalam representasinya, citra digital disusun dalam bentuk matriks berukuran $M \times N$,

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \dots & \dots & \dots & \dots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix}$$

di mana setiap elemen dalam matriks atau piksel, memiliki nilai intensitas tertentu. Nilai ini yang dikenal sebagai derajat keabuan (*grey level*) yang menentukan tingkat kecerahan pada piksel tersebut. Piksel dengan intensitas lebih tinggi akan tampak lebih terang dibandingkan yang memiliki intensitas lebih rendah. Struktur ini mendukung berbagai pengolahan citra digital, seperti kompresi, analisis, atau transformasi.

Gambar 2.1 Ilustrasi Piksel pada Gambar
(Sumber: [3])

Berdasarkan kombinasi warna pada piksel, citra dibagi menjadi tiga jenis yaitu citra biner, citra *grayscale*, citra RGB.

2.1.1. Citra Biner

Citra biner adalah jenis citra digital yang hanya memiliki dua nilai intensitas warna, yaitu 0 yang merepresentasikan warna hitam dan 1 yang merepresentasikan warna putih. Citra ini memiliki kedalaman bit sebesar 1 bit, sehingga cukup sederhana namun sangat bermanfaat dalam berbagai aplikasi, terutama pada proses segmentasi citra.

Untuk menghasilkan citra biner dari citra berwarna, diperlukan langkah awal berupa konversi citra berwarna menjadi citra *grayscale*. Setelah itu, ditentukan nilai ambang batas (*threshold*) yang berfungsi sebagai pemisah nilai intensitas. Jika nilai intensitas suatu piksel sama dengan atau lebih besar dari *threshold*, maka piksel tersebut dikonversi menjadi 1. Sebaliknya, jika nilai intensitasnya di bawah *threshold*, maka piksel tersebut dikonversi menjadi 0.

Gambar 2.1.1 Contoh Citra Biner
(Sumber: [4])

2.1.2. Citra *Grayscale*

Citra *grayscale* adalah citra digital yang merepresentasikan nilai intensitas piksel berdasarkan derajat keabuan tanpa melibatkan informasi warna lainnya. Dalam citra *grayscale* dengan kedalaman 8 bit, terdapat 256 tingkatan keabuan yang dimulai dari nilai 0 sebagai hitam sempurna hingga nilai 255 sebagai putih sempurna. Citra grayscale sering digunakan dalam analisis citra digital karena menyederhanakan informasi visual menjadi intensitas tunggal, sehingga lebih mudah untuk diolah dan dianalisis dalam berbagai aplikasi pemrosesan citra. dibuka di berbagai perangkat dan sistem operasi.

Citra RGB dapat dikonversi menjadi citra *grayscale* melalui persamaan berikut.

$$y = 0.22989 * R + 0.5870 * G + 0.1140 * B$$

Gambar 2.1.2 Contoh Citra *Grayscale*
(Sumber: [5])

2.1.3. Citra RGB

Citra RGB adalah citra digital yang terdiri dari tiga kanal warna utama, yaitu merah (*Red*), hijau (*Green*), dan biru (*Blue*), yang secara bersama-sama membentuk model warna RGB. Setiap kanal warna ini memiliki kedalaman bit sebesar 8-bit, yang memungkinkan setiap kanal untuk merepresentasikan 256 tingkat intensitas warna, mulai dari 0 hingga 255. Kombinasi intensitas dari ketiga kanal ini menghasilkan berbagai warna yang membentuk citra berwarna.

Pada masing-masing kanal, nilai 255 merepresentasikan warna sempurna dari kanal tersebut, sedangkan nilai 0 menunjukkan warna hitam. Sebagai contoh, pada kanal merah, nilai 255 mewakili warna merah sempurna, dan pada kanal hijau serta biru, nilai 255 masing-masing mewakili warna hijau dan biru sempurna. Penggabungan berbagai nilai intensitas pada ketiga kanal ini memungkinkan terciptanya berbagai warna yang lebih kompleks pada citra RGB. Model warna RGB ini sangat penting dalam pengolahan citra digital, karena memberikan representasi warna yang kaya dan fleksibel, yang banyak digunakan dalam aplikasi seperti pengolahan gambar, tampilan layar komputer, dan video.

Gambar 2.1.3 Contoh Citra RGB
(Sumber: Dokumen Penulis)

2.2. Kompresi Citra

Kompresi citra merupakan proses pengurangan ukuran file citra tanpa mengorbankan kualitas informasi yang terkandung di dalamnya. Tujuan utama dari kompresi citra adalah untuk mengurangi kebutuhan ruang penyimpanan dan mempercepat transmisi data melalui jaringan, terutama pada citra berukuran besar yang sering digunakan dalam berbagai aplikasi digital, seperti pengolahan citra medis, pengarsipan foto, dan sistem pengawasan video. Dalam kompresi citra, informasi yang tidak terlalu signifikan atau redundansi data dihilangkan atau disusun ulang untuk menghasilkan file dengan ukuran yang lebih kecil.

Metode pemampatan citra dapat dibedakan menjadi dua tipe utama, yaitu *lossy compression* dan *lossless compression*

2.2.1. *Lossy Compression*

Metode *lossy compression* adalah metode yang menghasilkan citra yang telah dipampatkan dengan ukuran file yang lebih kecil, meskipun terjadi kehilangan informasi selama proses pemampatan. Kehilangan data ini tidak terlalu mengganggu persepsi visual manusia, sehingga citra yang dipampatkan tetap tampak hampir identik dengan citra aslinya. Tujuan utama dari teknik ini adalah untuk mencapai nisbah pemampatan yang tinggi, sehingga ukuran file dapat diperkecil secara signifikan tanpa mengorbankan kualitas citra secara mencolok. Metode ini banyak digunakan dalam aplikasi di mana pengurangan ukuran file menjadi prioritas utama, seperti pada kompresi gambar untuk keperluan internet atau media sosial. Contoh dari metode *lossy compression* adalah *JPEG compression* dan *fractal image compression*.

2.2.2. *Lossless Compression*

Metode *lossless compression* adalah metode yang menghasilkan citra yang hasil pemampatannya identik dengan citra asli, baik dari segi data maupun kualitas visual, tanpa ada informasi yang hilang. Meskipun metode ini menghasilkan nisbah pemampatan yang lebih rendah dibandingkan dengan *lossy compression*, kualitas citra yang dihasilkan tetap terjaga dengan sempurna. Metode ini sangat dibutuhkan dalam aplikasi yang mengharuskan citra untuk tetap terjaga keasliannya, seperti pada citra medis atau citra X-ray, di mana kehilangan informasi dapat berakibat fatal. Beberapa contoh metode *lossless compression* adalah *Huffman coding*, *run-length encoding (RLE)*, dan *quantized coding*.

2.3. *Huffman Coding*

Huffman coding adalah salah satu metode pemampatan data yang termasuk dalam kategori *lossless compression*. Metode ini dikembangkan oleh David A. Huffman pada tahun 1952, dengan tujuan untuk menghasilkan pengkodean data yang efisien berdasarkan frekuensi kemunculan simbol-simbol dalam data. Pada dasarnya, Huffman coding menghasilkan representasi data yang lebih kecil tanpa mengorbankan kualitas data itu sendiri. Hal ini menjadikan *Huffman coding* banyak digunakan dalam berbagai aplikasi pemampatan data, seperti pemampatan teks, citra, dan file multimedia lainnya.

Pembuatan kode Huffman dimulai dengan membangun sebuah pohon Huffman, yang menjadi dasar dari algoritma ini. Pohon Huffman dibentuk dengan cara menggabungkan simbol-simbol yang memiliki frekuensi terendah menjadi satu simpul baru. Proses ini diulang hingga semua simbol bergabung menjadi satu pohon tunggal. Dalam pohon Huffman, setiap simbol diwakili oleh sebuah cabang, dan panjang cabang (kode biner) yang diberikan kepada setiap simbol berbanding terbalik dengan frekuensi kemunculannya. Simbol dengan frekuensi tinggi akan mendapatkan kode yang lebih pendek, sedangkan simbol dengan frekuensi rendah akan mendapatkan kode yang lebih panjang. Dengan demikian, Huffman coding memungkinkan pengkodean data yang lebih efisien.

Berikut adalah algoritma pembentukan Pohon Huffman:

1. Menghitung frekuensi kemunculan setiap simbol dalam data.
2. Menyusun simbol berdasarkan frekuensinya dari yang terendah hingga yang tertinggi.
3. Menggabungkan dua simbol dengan frekuensi terendah menjadi sebuah simpul baru, dan memasukkan simpul tersebut kembali ke dalam daftar.
4. Mengulangi langkah 3 hingga semua simbol tergabung menjadi satu pohon.
5. Memberikan label pada setiap sisi pohon secara konsisten, dengan sisi kiri diberi label 0 dan sisi kanan diberi label 1
6. Lintasan dari akar ke daun, yang berisi sisi-sisi pohon, akan membentuk deretan label yang menyatakan kode Huffman untuk simbol daun tersebut.

Kode biner yang dihasilkan dari algoritma metode *Huffman coding* bersifat *prefix-free*, yang berarti tidak ada kode yang merupakan awalan dari kode lainnya, sehingga proses *encoding* dan *decoding* dapat dilakukan dengan jelas dan tanpa ambiguitas. Dengan menggunakan metode *Huffman coding*, data dapat dipampatkan secara efektif tanpa kehilangan informasi, yang menjadikannya salah satu teknik pemampatan yang paling banyak digunakan dalam berbagai bidang, termasuk kompresi citra digital.

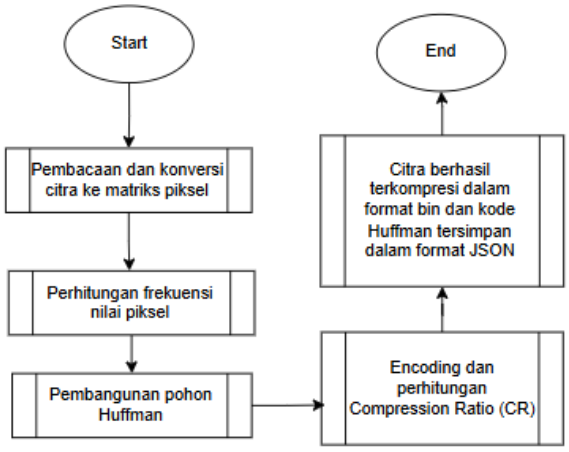
III. IMPLEMENTASI

3.1. Batasan Pembahasan

Pada makalah ini, implementasi algoritma Huffman dibatasi pada citra digital dengan format *grayscale*. Pemilihan citra *grayscale* didasarkan pada karakteristiknya yang hanya memiliki intensitas warna tunggal per piksel dengan rentang nilai 0 hingga 255. Hal ini memberikan kemudahan dalam analisis distribusi frekuensi intensitas piksel dibandingkan citra berwarna yang memiliki tiga kanal (RGB). Dengan demikian, pembahasan ini lebih fokus pada proses optimasi kompresi citra *grayscale* tanpa mempertimbangkan kompleksitas tambahan dari format citra berwarna. Selain itu, implementasi ini hanya membahas citra berukuran kecil hingga menengah, agar memudahkan simulasi proses kompresi dan dekompresi. Proses ini tidak mencakup aspek-aspek seperti penanganan metadata citra, *header* file, atau format penyimpanan gambar yang spesifik (seperti PNG atau JPEG).

3.2. Alur Kompresi Huffman dan Studi Kasus

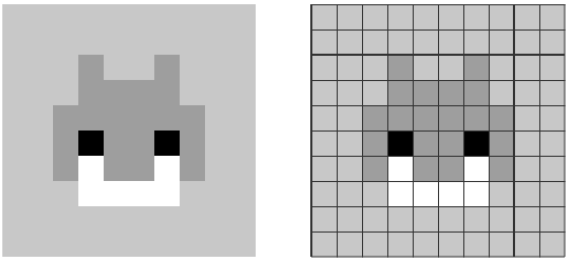
Kompresi data pada citra *grayscale* menggunakan algoritma Huffman diawali dengan analisis mendalam terhadap pola intensitas piksel dalam citra tersebut. Tujuan utama adalah mereduksi ukuran data tanpa kehilangan informasi yang diperlukan untuk merekonstruksi citra secara akurat. Berikut adalah alur kompresi citra *grayscale* menggunakan algoritma Huffman.



Gambar 3.2 Alur Kompresi Citra *Grayscale* Menggunakan Algoritma Huffman
(Sumber: Dokumen Penulis)

3.2.1. Konversi Citra ke Matriks Piksel

Citra *grayscale* yang akan dikompresi pada dasarnya dapat direpresentasikan sebagai matriks dua dimensi, di mana setiap elemen matriks merepresentasikan nilai intensitas piksel dengan rentang 0 hingga 255.



Gambar 3.2.1.1 Ilustrasi Piksel Pada Citra *Grayscale*
(Sumber: Dokumen Penulis)

Sebagai ilustrasi, citra *grayscale* berukuran 10×10 piksel yang ditunjukkan pada **Gambar 3.2.1.1** dapat digambarkan dalam matriks dua dimensi berikut.

$$\begin{bmatrix} 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 \\ 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 \\ 200 & 200 & 200 & 158 & 200 & 200 & 158 & 200 & 200 & 200 \\ 200 & 200 & 200 & 158 & 158 & 158 & 158 & 200 & 200 & 200 \\ 200 & 200 & 158 & 158 & 158 & 158 & 158 & 158 & 200 & 200 \\ 200 & 200 & 158 & 0 & 158 & 158 & 0 & 158 & 200 & 200 \\ 200 & 200 & 158 & 255 & 158 & 158 & 255 & 158 & 200 & 200 \\ 200 & 200 & 200 & 255 & 255 & 255 & 255 & 200 & 200 & 200 \\ 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 \\ 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 & 200 \end{bmatrix}$$

Dalam implementasi program, matriks dua dimensi ini diubah menjadi matriks satu dimensi untuk mempermudah pemrosesan data. Berikut implementasi fungsi pembacaan citra dan konversi citra ke matriks piksel.

Gambar 3.2.1.2 Implementasi Pembacaan dan Konversi Citra ke Matriks Piksel
(Sumber: Dokumen Penulis)

3.2.2. Perhitungan Frekuensi Nilai Piksel

Setelah matriks satu dimensi diperoleh, langkah berikutnya adalah menghitung jumlah kemunculan setiap nilai intensitas dalam matriks tersebut. Proses ini menghasilkan tabel frekuensi yang menjadi dasar pembangunan pohon Huffman. Berdasarkan contoh data di atas, tabel frekuensi nilai intensitas pikselnya adalah sebagai berikut.

Simbol/Nilai	0	158	200	255
Frekuensi	2	20	72	6

Berikut implementasi fungsi perhitungan frekuensi nilai piksel.

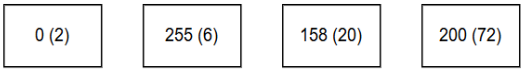
```
def calculate_frequencies(self, pixel_array):
        frequencies = defaultdict(int)
        for pixel in pixel_array:
            frequencies[pixel] += 1
        return frequencies
```

Gambar 3.2.2 Implementasi Perhitungan Frekuensi Nilai Piksel
(Sumber: Dokumen Penulis)

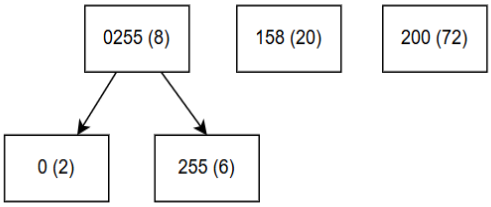
3.2.3. Pembentukan Pohon Huffman

Setelah frekuensi kemunculan setiap simbol dihitung, maka akan dibentuk pohon Huffman dengan langkah-langkah sebagai berikut.

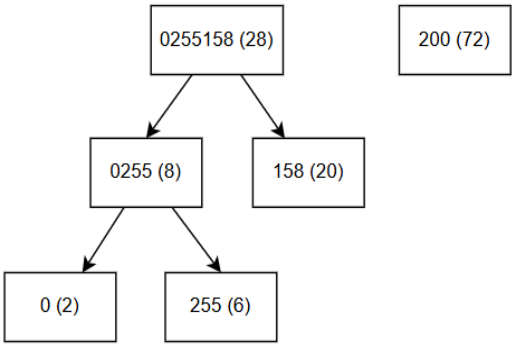
1. Simbol diurutkan berdasarkan frekuensi kemunculannya dari kecil ke besar.



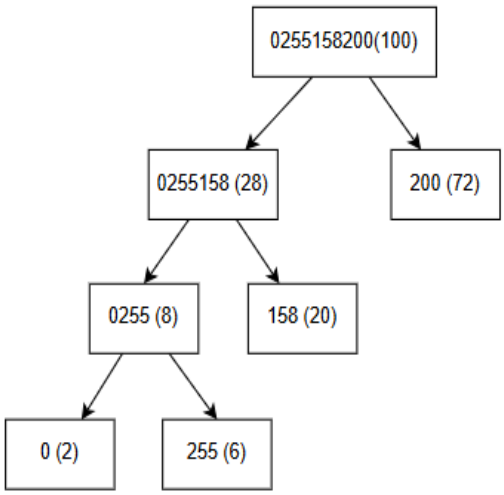
2. Simbol 0 dan 255 digabungkan menjadi 0255 dengan frekuensi 0255 = 2+6 = 8, dan simbol baru 0255 ditempatkan dalam urutan yang terurut.



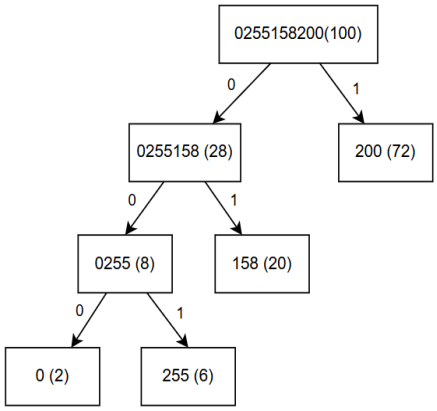
3. Simbol 0255 dan 158 digabungkan menjadi 0255158 dengan frekuensi 0255158 = 8+20 = 28, dan simbol baru 0255158 ditempatkan dalam urutan yang terurut.



4. Simbol 0255158 dan 200 digabungkan menjadi 0255158200 dengan frekuensi 0255158200 = 28+72 = 100



5. Simbol sudah habis dan pohon Huffman sudah berhasil terbentuk. Kemudian, sisi-sisi kiri dalam pohon Huffman diberi label 0, dan sisi-sisi kanan diberi label 1.



6. Pohon ditelusuri dari akar ke daun, dan lintasan dari akar ke daun berisi rangkaian bit yang menyatakan kode Huffman untuk setiap simbol di dalam daun. Berikut tabel frekuensi dan kode Huffman yang berhasil dibentuk.

Simbol/Nilai	Frekuensi	Kode Huffman
0	2	000
158	20	01
200	72	1
255	6	001

Dapat dilihat bahwa Kode Huffman yang dihasilkan bersifat *prefix-free*, yang berarti tidak ada kode yang merupakan awalan dari kode lainnya, sehingga proses *encoding* dan *decoding* dapat dilakukan dengan jelas dan tanpa ambiguitas.

Berikut implementasi fungsi pembentukan pohon Huffman.

```python
class HuffmanNode:
    def __init__(self, value=None, freq=0):
        self.value = value
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq


class ImageCompressor:
    def __init__(self):
        self.huffman_codes = {}
        self.huffman_tree = None
```

```python
def build_huffman_tree(self, frequencies):
    heap = []
    for value, freq in frequencies.items():
        node = HuffmanNode(value, freq)
        heapq.heappush(heap, node)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)

        internal_node = HuffmanNode(freq=left.freq + right.freq)
        internal_node.left = left
        internal_node.right = right

        heapq.heappush(heap, internal_node)

    self.huffman_tree = heap[0]

def generate_huffman_codes(self, node=None, code=""):
    if node is None:
        node = self.huffman_tree

    if node.value is not None:
        self.huffman_codes[node.value] = code
        return

    self.generate_huffman_codes(node.left, code + "0")
    self.generate_huffman_codes(node.right, code + "1")
```

Gambar 3.2.3 Implementasi Pembentukan Pohon Huffman
(Sumber: Dokumen Penulis)

3.2.4. Proses *Encoding*

Setiap nilai intensitas piksel dalam matriks awal dikodekan menggunakan kode Huffman yang telah dibentuk. Matriks awal dikonversi menjadi rangkaian bit biner sebagai berikut:

```
11111111111111111111111011101111111010101011
11110101010101011111010000101000011111010010
10100101111110010010010011111111111111111111
1111
```

Proses *encoding* ini menghasilkan data terkompresi yang siap untuk disimpan dalam format file biner (.bin). File biner ini hanya menyimpan rangkaian bit hasil *encoding*, sementara kode Huffman disimpan terpisah dalam format JSON untuk mempermudah proses dekompresi.

Berikut implementasi fungsi *encoding* dan penyimpanan kode Huffman dalam format JSON.

```python
def encode_image(self, pixel_array):
    encoded_data = ""
    for pixel in pixel_array:
        encoded_data += self.huffman_codes[pixel]
    return encoded_data

def save_compressed(self, encoded_data, codes, output_path):
    padded_length = 8 - (len(encoded_data) % 8)
    encoded_data += "0" * padded_length

    byte_array = bytearray()
    for i in range(0, len(encoded_data), 8):
        byte = encoded_data[i:i+8]
        byte_array.append(int(byte, 2))

    with open(output_path + ".bin", "wb") as f:
        f.write(bytes([padded_length]))
        f.write(byte_array)

    with open(output_path + "_codes.json", "w") as f:
        json.dump({str(k): v for k, v in codes.items()}, f)
```

Gambar 3.2.4 Implementasi Proses *Encoding* dan Penyimpanan Kode Huffman dalam Format JSON
(Sumber: Dokumen Penulis)

3.2.5. Perhitungan *Compression Ratio*

Untuk mengevaluasi efisiensi kompresi, ukuran data sesudah dan sebelum kompresi dibandingkan untuk memperoleh nilai *Compression Ratio* (CR) atau nisbah pemampatan dengan rumus sebagai berikut:

$$Compression\ Ratio\ (CR) = \frac{S_c}{S_o} \times 100\%$$

Keterangan:
S_o : Ukuran file asli (*Original Size*)
S_c : Ukuran file hasil kompresi (*Compressed Size*).

Diasumsikan setiap nilai intensitas direpresentasikan dalam 8 bit, dan terdapat 100 nilai intensitas pada gambar. Dengan demikian, ukuran data awal (S_o) adalah:

$$S_o = 8\ bit\ \times 100 = 800\ bit$$

Ukuran data hasil kompresi (S_c), yang telah dikodekan dengan kode Huffman, dihitung sebagai berikut:

$$S_c = (2 \times 3) + (20 \times 2) + (72 \times 1) + (6 \times 3) = 136\ bit$$

Dengan demikian, *Compression Ratio* dihitung sebagai berikut:

$$CR = \frac{S_c}{S_o} \times 100\% = \frac{136\ bit}{800\ bit} \times 100\% = 17\%$$

Berdasarkan hasil tersebut, citra berhasil dimampatkan menjadi 17% dari citra semula atau 83% dari citra semula telah dimampatkan. Berikut implementasi fungsi perhitungan *Compression Ratio*.

```
def calculate_compression_ratio(self, original_size, compressed_size):
        return compressed_size/original_size * 100
```

Gambar 3.2.5 Implementasi Perhitungan *Compression Ratio*
(Sumber: Dokumen Penulis)

3.2.6. Proses Dekompresi

Untuk mengembalikan citra ke bentuk aslinya, file biner dibaca dan dikonversi kembali menjadi nilai intensitas piksel menggunakan kode Huffman yang telah disimpan dalam file JSON. Proses ini memastikan rekonstruksi citra yang identik dengan data awal. Berikut implementasi fungsi dekompresi.

```
def decode_image(self, input_path, codes_path, output_path, original_size):
        with open(codes_path, 'r') as f:
            codes = json.load(f)
            codes = {int(k): v for k, v in codes.items()}

        with open(input_path, 'rb') as f:
            padding_length = int.from_bytes(f.read(1), byteorder='big')
            byte_data = f.read()

        binary_data = ""
        for byte in byte_data:
            binary_data += format(byte, '08b')
        binary_data = binary_data[:-padding_length] if padding_length > 0 else binary_data

        reverse_codes = {v: k for k, v in codes.items()}
        current_code = ""
        decoded_pixels = []

        for bit in binary_data:
            current_code += bit
            if current_code in reverse_codes:
                decoded_pixels.append(reverse_codes[current_code])
                current_code = ""

        decoded_array = np.array(decoded_pixels, dtype=np.uint8)
        decoded_image = Image.fromarray(decoded_array.reshape(original_size[::-1]))
        decoded_image.save(output_path)
```

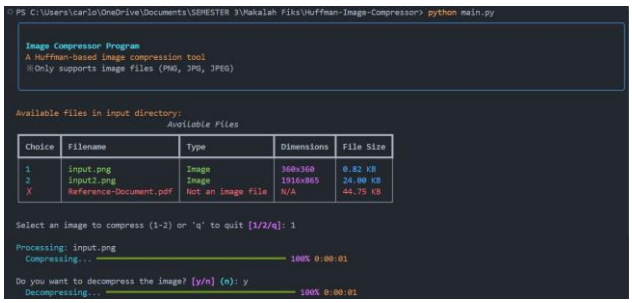
Gambar 3.2.6 Implementasi Proses Dekompresi
(Sumber: Dokumen Penulis)

IV. HASIL DAN PEMBAHASAN

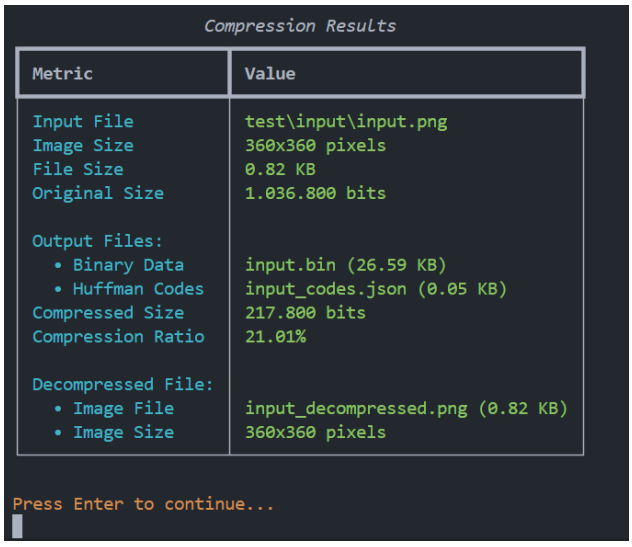
Pengujian algoritma Huffman dilakukan untuk menganalisis kinerja kompresi dan dekompresi pada citra digital. Dalam pengujian ini, citra input dengan format *grayscale* dikompresi menjadi file dengan format .bin, dan hasil dekompresinya dibandingkan dengan citra input.

Gambar 4.1 Citra Grayscale Pengujian
(Sumber: Dokumen Penulis)

Pada pengujian ini, digunakan citra input *grayscale* berukuran 360×360 piksel yang ditunjukkan pada **Gambar 4.1**. Hasil menunjukkan efisiensi algoritma Huffman dalam mengurangi redundansi data tanpa kehilangan informasi penting.



Gambar 4.2 Proses Pengujian Kompresi dan Dekompresi
(Sumber: Dokumen Penulis)



Gambar 4.3 Hasil Pengujian Kompresi dan Dekompresi
(Sumber: Dokumen Penulis)

Gambar 4.4 Kode Huffman Hasil Pengujian
(Sumber: Dokumen Penulis)

Ukuran file .bin hasil kompresi ternyata lebih besar dibandingkan ukuran file citra input. Seperti yang terlihat pada **Gambar 4.3** bahwa ukuran file citra input sebesar 0.82 kb, sedangkan ukuran file .bin hasil kompresi sebesar 26.59 kb. Hal ini disebabkan oleh beberapa faktor teknis dalam proses kompresi. Data hasil *encoding* Huffman disesuaikan agar panjangnya merupakan kelipatan 8 bit dengan menambahkan *padding* pada akhir data biner. Penambahan ini dilakukan untuk memastikan kompatibilitas dengan format byte, namun menyebabkan peningkatan ukuran file. Selain itu, *byte* pertama dalam file .bin digunakan untuk menyimpan informasi panjang *padding* yang juga menambah ukuran file.

Hasil pengujian dekompresi menunjukkan bahwa citra yang dihasilkan identik dengan citra input dalam hal ukuran matriks piksel dan intensitasnya seperti ditunjukkan pada **Gambar 4.3**. Hal ini membuktikan bahwa algoritma Huffman bersifat *lossless*, yang berarti tidak ada informasi data piksel yang hilang selama proses

kompresi dan dekompresi. Namun, perbedaan kecil ditemukan pada ukuran file citra hasil dekompresi dibandingkan file citra input. Hal ini terjadi karena metadata yang terdapat pada file gambar asli, seperti informasi perangkat atau waktu pembuatan, tidak dikelola atau diproses oleh algoritma Huffman. Oleh karena itu, meskipun hasil dekompresi identik secara visual, file hasil dekompresi tidak membawa metadata dari file input.

Struktur kode Huffman memiliki sifat *prefix-free* seperti ditunjukkan pada **Gambar 4.4**, yang memastikan bahwa setiap kode biner yang dihasilkan tidak menjadi awalan bagi kode lainnya. Sifat ini memungkinkan proses pengkodean dan penguraian data dilakukan secara efisien tanpa risiko ambiguitas dalam interpretasi data. Setiap nilai intensitas piksel pada citra digital diberikan representasi kode biner yang unik, yang panjangnya ditentukan berdasarkan frekuensi kemunculan nilai tersebut. Nilai dengan frekuensi kemunculan tinggi diberikan kode yang lebih pendek, sementara nilai dengan frekuensi rendah mendapatkan kode yang lebih panjang. Dengan demikian, algoritma ini mampu mengurangi ukuran data secara signifikan, terutama pada citra dengan distribusi intensitas piksel yang tidak merata. Pada citra dengan area dominan warna seragam, seperti latar belakang, algoritma ini menghasilkan kode pendek untuk nilai yang sering muncul, sehingga meningkatkan efisiensi kompresi. Sebaliknya, pada citra dengan variasi intensitas tinggi, pohon Huffman yang dihasilkan akan lebih kompleks untuk mencerminkan keragaman data. Meskipun demikian, proses dekompresi tetap dapat dilakukan dengan baik karena struktur kode Huffman mempermudah pemetaan setiap kode biner ke nilai aslinya tanpa memerlukan pemisah tambahan. Fleksibilitas algoritma Huffman menjadikannya tidak hanya relevan untuk kompresi citra digital, tetapi juga untuk data lain seperti teks atau file multimedia, sehingga dapat memenuhi berbagai kebutuhan pengolahan data secara efisien.

V. KESIMPULAN DAN SARAN

Berdasarkan hasil pengujian, algoritma Huffman memiliki keunggulan dalam mereduksi ukuran data secara signifikan pada citra dengan distribusi intensitas piksel yang tidak merata. Namun, algoritma ini kurang optimal untuk citra dengan distribusi intensitas merata karena pengurangan ukuran data yang dihasilkan menjadi terbatas. Kelemahan lain dari algoritma Huffman adalah ukuran file hasil kompresi yang lebih besar dibandingkan file asli akibat penambahan padding untuk memastikan panjang rangkaian bit biner merupakan kelipatan 8 agar sesuai dengan format *byte*. Untuk mengatasi kelemahan tersebut, diperlukan alternatif penyimpanan lain selain format file biner untuk mengurangi *overhead* yang dihasilkan. Selain itu, algoritma Huffman dapat dikombinasikan dengan algoritma seperti *Run-Length Encoding* (RLE) untuk meningkatkan efisiensi pada citra dengan area dominan warna seragam.

VI. LAMPIRAN

Program lengkap dari makalah ini dapat diakses melalui link repository github berikut:
https://github.com/carllix/Huffman-Image-Compressor

Video penjelasan dari makalah ini dapat diakses melalui link berikut:
https://youtu.be/oYxnImYZzOA?si=SpeHvIH_X56ahWly

VII. UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya yang melimpah dalam proses penulisan makalah ini, sehingga dapat diselesaikan tepat waktu. Penulis juga mengucapkan terima kasih kepada Dr. Ir. Rinaldi Munir, M.T., selaku dosen mata kuliah IF1220 Matematika Diskrit Kelas K01, yang telah memberikan bimbingan dan ilmu pengetahuan yang sangat berharga, serta telah menyediakan *website* sebagai sumber belajar untuk mata kuliah Matematika Diskrit, yang sangat membantu dalam memahami materi dengan lebih baik. Selain itu, penulis juga menyampaikan terima kasih yang sebesar-besarnya kepada kedua orang tua yang selalu memberikan dukungan, semangat, dan doa yang tak terhingga, yang menjadi sumber motivasi dan kekuatan dalam setiap langkah penulis.

REFERENSI

[1] Munir, Rinaldi. 2024. “Pohon (Bagian 2)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/24-Pohon-Bag2-2024.pdf (Diakses pada 30 Desember 2024)

[2] Munir, Rinaldi. 2024. “Pemampatan Citra (Bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2024-2025/25-Image-Compression-Bagian1-2024.pdf (Diakses pada 30 Desember 2024).

[3] Munir, Rinaldi. 2024. “Pengantar Pemrosesan Citra Digital (Bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2024-2025/01-Pengantar-Pemrosesan-Citra-Digital-Bag1-2024.pdf (Diakses pada 30 Desember 2024).

[4] Munir, Rinaldi. 2024. “Format Citra”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2024-2025/04-Format-citra-2024.pdf (Diakses pada 30 Desember 2024).

[5] Pemrograman Matlab. 2022. “Pengolahan Citra”. https://pemrogramanmatlab.com/pengolahan-citra-digital/ (Diakses pada 30 Desember 2024).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Januari 2025

Carlo Angkisan
13523091