Optimasi Memori dalam Pengiriman Audio pada Aplikasi Chat dengan Kode Huffman

Debrina Veisha Rashika Wijayanto - 13522025[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522025@std.stei.itb.ac.id

***Abstrak*— Kompresi adalah proses perubahan ukuran data menjadi lebih kecil yang memiliki peran penting dalam memfasilitasi pengiriman dan penyimpanan data, terutama dalam aplikasi seperti *chatting*. Kompresi dapat dibagi menjadi dua jenis, yaitu, *lossless* dan *lossy*. Salah satu contoh teknik *lossless* adalah *Huffman Coding*. Algoritma *Huffman Coding* menganalisis frekuensi kemunculan simbol-simbol dalam data untuk melakukan proses kompresi. Penelitian ini bertujuan untuk meneliti dampak kompresi Huffman Coding pada pengiriman audio, baik dari segi efektivitas maupun efisiensi.**

***Keywords*— Audio, *Huffman Coding*, Kompresi**

I. PENDAHULUAN

Seiring berkembangnya teknologi, komunikasi jarak jauh antar sesama manusia menjadi lebih mudah. Terlebih lagi banyak aplikasi yang menyediakan berbagai fitur menarik yang memudahkan manusia untuk tetap saling berinteraksi tanpa batasan jarak. Salah satu fitur yang disediakan oleh aplikasi komunikasi modern dan sering digunakan untuk berinteraksi adalah fitur pesan suara. Dengan adanya fitur pesan suara pengguna dapat langsung mengirimkan pesan dengan merekam suara mereka tanpa harus mengetik terlebih dahulu. Cara ini dinilai lebih cepat dan efisien, karena tidak membuang waktu untuk mengetik dan pesan yang ingin disampaikan menjadi lebih jelas dibanding jika harus menulis pesan secara tertulis.

Proses pengiriman pesan suara melibatkan transformasi sinyal audio menjadi sinyal digital. Sinyal analog didigitalisasi melalui pengambilan sampel dengan periode pengambilan sampel tetap. Oleh karena itu, total data yang disimpan menjadi besar. Karena keterbatasan kecepatan transmisi atau kapasitas penyimpanan yang dimiliki, sinyal digital ini harus dikompresi secara efisien.

Kecepatan pengiriman bergantung dengan ukuran informasi yang akan dikirimkan. Salah satu solusi dari permasalahan ini adalah dengan menjalankan kompresi sebelum data dikirimkan. Setelah itu, data akan didekompresi oleh penerima untuk mendapatkan hasil data yang memiliki ukuran memori lebih rendah.

Teknik kompresi sangat penting untuk menghasilkan data hasil yang baik. Karena ukuran data yang besar akan memerlukan proses yang lebih kompleks. Teknik kompresi terbagi menjadi dua yaitu, *lossless* dan *lossy* [1]. Kompresi *lossy* mengurangi jumlah frekuensi yang dimiliki oleh audio asli, namun masih menjaga frekuensi suara yang masih dapat didengar oleh manusia. Sedangkan, kompresi *lossless* akan memperkecil ukuran memori tanpa mengurangi isi dari data dengan menggunakan aproksimasi linier dan pemrograman entropi. Melalui kedua Teknik kompresi ini data akan dialokasikan untuk meningkatkan kecepatan dalam melakukan transmisi data.

Teknik *lossless* dapat diterapkan pada file WAV dengan menggunakan algoritma Huffman. Algoritma Huffman mengelompokkan dan mengonversi frekuensi simbol-simbol dalam data audio, meminimalkan ukuran data tanpa mengorbankan kualitas suara. Algoritma ini menghitung frekuensi dari masing-masing simbol dan akan dikoversi menjadi rangkaian kode bit yang lebih pendek. Dalam konteks ini, makalah ini akan menjelaskan secara lebih rinci tentang teknik kompresi, fokus pada pendekatan *lossless* dengan algoritma Huffman pada file WAV.

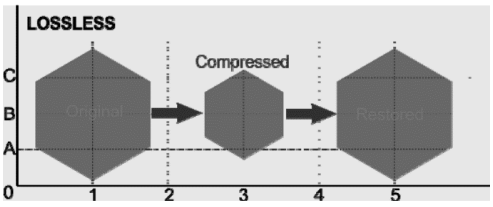
II. TEORI DASAR

A. Kompresi Data

Kompresi adalah perubahan ukuran suatu data menjadi lebih kecil dari ukuran aslinya [2]. Hal ini bertujuan untuk mempermudah dalam proses pengiriman dan penyimpanan data. Kompresi data dapat dijalankan dengan dua tipe sebagai berikut [1].

- Kompresi *Lossless* adalah kelas dari kompresi data yang memungkinkan data asli untuk disusun kembali dari data hasil kompresi. Teknik ini digunakan untuk tetap menjaga kondisi asli data.
- Kompresi *Lossy* menghapus bagian dari data yang tidak penting atau tidak digunakan. Teknik ini menghasilkan hasil data dengan ukuran yang kecil dibandingkan dengan kompresi *Lossless.*

Dengan adanya kedua tipe kompresi ini, aplikasi yang membutuhkan pemeliharaan keaslian data dapat memilih kompresi *lossless*, sementara aplikasi yang bersifat lebih fleksibel terhadap kehilangan informasi dapat memilih kompresi *lossy*.



Gambar 2.1. Ilustrasi Kompresi Lossless
(Sumber: Comparison of Lossless Compression, Hidayat)

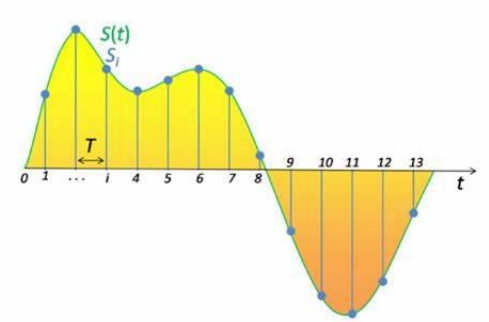
B. Digitalisasi Sinyal Audio

Bunyi yang biasa terdengar dalam kehidupan sehari-hari berbentuk gelombang analog. Gelombang ini dieredam oleh tekanan udara yang ada di sekitar sampai akhirnya bisa didengar oleh manusia. Namun, dalam dunia digital, audio perlu diubah menjadi bentuk yang dapat dipahami dan diolah oleh perangkat elektronik. Audio digital melibatkan pemecahan bunyi menjadi bagian-bagian kecil yang dapat diolah lebih lanjut. Proses ini dikenal sebagai digitalisasi atau *sampling* [3].

Dalam digitalisasi, tegangan audio diukur pada titik-titik tertentu dalam interval waktu tertentu, dan hasilnya diinterpretasikan sebagai data numerik. Setelah itu, data ini dapat disimpan dalam bentuk file digital. Proses *sampling* memainkan peran kunci dalam digitalisasi. Ini melibatkan pengukuran tegangan audio pada titik-titik diskrit pada interval waktu tertentu. Semakin tinggi tingkat sampling, semakin baik representasi digital dari sinyal audio.

Kelebihan dari audio digital adalah produksi suara yang dihasilkan lebih jernih dan berkualitas. Dengan kualitas audio yang baik, dapat dilakukan pengolahan atau produksi sinyal audio secara berulang tanpa mengurangi kualitas suara asli dari audio tersebut. Meski begitu, jika audio memiliki kapasitas yang besar, diperlukan proses kompresi yang sesuai untuk memperkecil ukuran dengan tetap menjaga kualitas dari audio tersebut.

Dengan kualitas audio yang ditingkatkan, audio digital menjadi fondasi yang penting dalam berbagai aplikasi, termasuk komunikasi jarak jauh, produksi musik, dan pengembangan teknologi audio. Kompresi audio merupakan langkah penting untuk mengatasi kendala ukuran file dalam lingkungan digital yang serba cepat dan efisien.



Gambar 2.2. Proses Sampling pada Audio Digital
(Sumber: www.hackers-arise.com)

C. Pohon

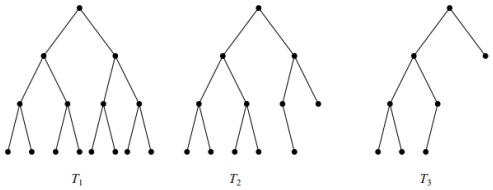
Pohon merupakan sebuah graf tak-berarah di mana setiap simpul saling terhubung tanpa membentuk sirkuit [4]. Pohon dapat memiliki variasi tertentu tergantung pada sifat dan arah hubungan antar simpulnya.

- Pohon Terurut (Ordered Tree):

Pohon terurut adalah pohon yang memiliki urutan atau aturan tertentu dalam penataan anak-anaknya. Dalam konteks ini, urutan simpul dan anak-anaknya menjadi penting, sehingga dapat memberikan struktur hierarkis yang dapat diikuti.

- Pohon Biner:

Pohon biner adalah jenis pohon terurut di mana setiap simpul memiliki maksimal dua anak, yaitu anak kiri dan anak kanan. Pohon biner sering digunakan dalam implementasi struktur data seperti pohon pencarian biner *(binary search tree)*.



Gambar 2.3. Ilustrasi Pohon Biner
(Sumber: Matematika Diskrit Jilid 3 2010, Rinaldi Munir)

D. Huffman Coding

Huffman Coding merupakan algoritma kompresi data yang bersifat *lossless* yang mengubah data menjadi prefiks optimal [5]. Kode prefix adalah himpunan kode yang didalamnya tidak ada anggota kumpulan yang menjadi awalan dari anggota himpunan lainnya. Konversi data menjadi kode prefiks dengan *Huffman Coding* bertujuan agar tidak menimbulkan ambiguitas saat dilakukan proses pemulihan kode menjadi data awal.

Algoritma *Huffman Coding* melihat frekuensi kemunculan simbol-simbol dalam data untuk melakukan proses kompresi. Simbol-simbol dengan frekuensi kemunculan rendah akan dikonversi menjadi kode bit yang lebih panjang dibandingkan dengan simbol-simbol yang memiliki frekuensi kemunculan lebih tinggi. Proses konversi menggunakan *Huffman Coding* mengadaptasi konsep dari pohon biner. Langkah-langkah pembentukan Kode Huffman, sebagai berikut [4]:

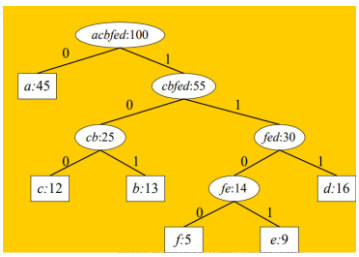
1) Pilih dua simbol dengan peluang kemunculan (*probability*) paling kecil.
2) Setelah itu, pilih dua simbol lainnya, termasuk simbol baru, yang mempunyai peluang terkecil.
3) Ulangi langkah 1 dan 2 sampai seluruh simbol habis.
4) Beri label secara konsisten, misalnya, sisi kiri dengan 0 dan sisi kanan dengan 1.
5) Lintasan dari akar ke daun berisi sisi-sisi pohon yang deretan labelnya menyatakan kode Huffman untuk simbol daun tersebut.

Contohnya terdapat sebuah pesan sepanjang 100 karakter yang tersusun oleh huruf a, b, c, d, e, dan f dengan frekuensi kemunculan sebagai berikut:

Karakter	a	b	c	d	e	f
Frekuensi	45	13	12	16	9	5

Tabel 2.1. Frekuensi Kemunculan Huruf

Dengan mengikuti langkah-langkah pembentukan kode Huffman, dapat dibentuk sebuah pohon biner sebagai berikut:



Gambar 2.4. Pohon Biner Hasil Kode Huffman
(Sumber: Matematika Diskrit Jilid 3 2010, Rinaldi Munir)

Dari pohon biner diatas maka didapat Kode Huffman untuk masing-masing huruf yang dapat dilihat pada tabel berikut:

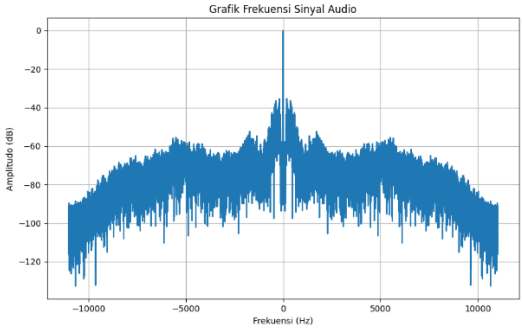
Huruf	Kode Huffman	Huruf	Kode Huffman
a	0	d	111
b	101	e	1101
c	100	f	1100

Tabel 2.2. Kode Huffman untuk Setiap Karakter

E. Teknik Kompresi Lossless pada File WAV

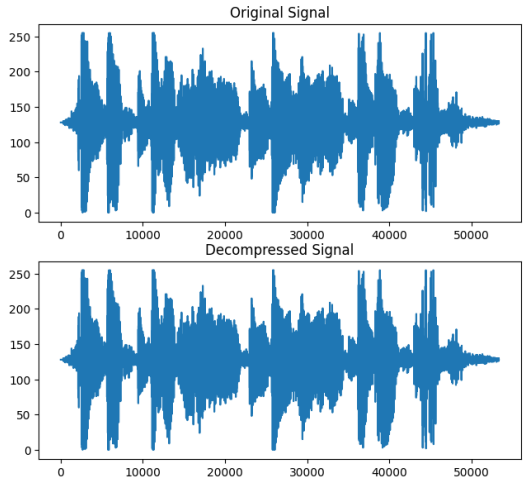
WAV adalah format audio *lossless* yang tidak mengkompresi rekaman audio analog dari asalnya. WAV menyediakan kecepatan sampel dan kedalaman bit yang sangat tinggi. Hal ini memungkinan untuk memberikan semua frekuensi yang didengar oleh manusia. File WAV menggunakan Teknik *Pulse Code Modulation* (PCM) untuk mangambil sampel gelombang analog di setiap intervalnya. PCM memetakan amplitudo sinyal audio pada setiap titik waktu menjadi nilai digital yang terkait, menghasilkan representasi akurat dari gelombang aslinya.

PCM mengkonversi data analog menjadi digital dalam berbagai tingkatan Sebagai contoh, standar PCM untuk file audio CD adalah 44.1kHz dan 16 bit. Ini berarti bahwa dalam satu detik, 44,100 sampel audio diambil, dan setiap sampel direpresentasikan dengan 16 bit. Frekuensi sampel yang tinggi seperti ini memungkinkan untuk mereproduksi rentang frekuensi yang luas. Namun, untuk keperluan audiofil, pengaturan sampel stereo dapat ditingkatkan hingga 48kHz dan 24 bit karena memberikan lebih banyak detail pada frekuensi tinggi dan rendah untuk dimasukkan ke dalam campuran audio dengan kualitas premium [6].



Gambar 2.5 Grafik Frekuensi Sinyal Audio

Dari informasi yang ada pada *header* file WAV bisa didapat sampel audio yang merupakan nilai amplitudo yang merepresentasikan getaran suara pada setiap titik waktu dalam file audio. Dalam file WAV dengan format PCM, nilai-nilai amplitudo direpresentasikan sebagai angka-angka dalam rentang tertentu, misalnya, dari -32768 hingga 32767 untuk sampel audio 16-bit. Selain itu, dari *header* juga bisa didapat informasi frekuensi sinyal audio tersebut. Pada grafik bisa dilihat bahwa nilai dari amplitudo sebuah audio dapat muncul beberapa kali pada dua atau lebih titik. Probabilitas kemunculan dari amplitudo dapat digunakan dalam penentuan Kode Huffman dalam proses kompresi audio. Kode Huffman memanfaatkan distribusi probabilitas untuk menghasilkan representasi biner yang efisien dari nilai amplitudo, meminimalkan ukuran file tanpa kehilangan informasi esensial.



Gambar 2.6. Perbandingan Grafik Sebelum dan Sesudah Dilakukan Kompresi dengan Kode Huffman

Setelah melakukan kompresi dengan kode Huffman, lalu medekompresinya kembali, terlihat pada grafik di atas tidak terdapat perubahan sinyal yang berarti kualitas dari audio tetap terjaga. Meski begitu, jumlah memori yang terpakai telah berkurang setelah melakukan kompresi audio. Pendekatan ini memungkinkan penyimpanan dan transmisi audio yang lebih efisien dan bermanfaat terutama pada situasi dengan keterbatasan *bandwidth* atau ruang penyimpanan.

F. Aplikasi Praktis pada Pengiriman Audio

Batas pengiriman audio melalui aplikasi pesan instan seperti WhatsApp, Line, dan aplikasi serupa bervariatif bergantung pada kebijakan dan batasan masing-masing *platform*. Namun, pada umumnya, aplikasi pesan instan memiliki batasan ukuran file untuk pengiriman pesan.

Gambar 2.7. Berbagai Aplikasi Chatting
(Sumber: www.droila.com)

Contohnya, pada aplikasi WhatsApp, batas ukuran file untuk pengiriman adalah sekitar 16 MB pada *platform* Android dan sekitar 12 MB pada *platform* iOS [7]. Line dan aplikasi lain juga memiliki batasan yang serupa.

Untuk mengatasi batasan ini, seringkali dilakukan kompresi file audio sebelum dikirim. Dengan mengompresi file audio, ukuran file dapat dikurangi tanpa mengorbankan kualitas suara secara signifikan. Kompresi audio menjadi penting karena tidak hanya mengoptimalkan penggunaan *bandwidth*, tetapi juga berdampak positif pada kecepatan transmisi data dan umur baterai perangkat [8]. Pemahaman tentang batasan ini menjadi penting dalam merancang solusi praktis dalam pengiriman pesan berupa audio.

III. IMPLEMENTASI

Pada bagian ini akan dijelaskan mengenai kode program yang digunakan untuk melakukan kompresi pada audio dengan menggunakan Kode Huffman. Program ini menggunakan Bahasa python.

A. Program Utama

```python
if __name__ == '__main__':
    # membaca file audio
    freq, original_signal = wavfile.read('./StarWars3.wav')
    # melakukan kompresi pada file
    Compress = Compressor(original_signal)
    compressed_signal, code = Compress.maincompressor()

    # melakukan dekompress pada file yang telah dikompresi
    DecompressorObject = Decompressor(compressed_signal, code)
    decompressed_signal = DecompressorObject.decompressor()

    # Menampilkan grafik sinyal asli dan sinyal yang telah di-decompress
    plot_signals(original_signal, decompressed_signal)

    # Menghitung ukuran file sebelum kompresi
    original_size = os.path.getsize('./StarWars3.wav') * 8  # dikali 8 karena dalam bit
    print(f'Ukuran File Sebelum Kompresi: {original_size} bits')

    # Menghitung ukuran file setelah kompresi
    compressed_size = len(''.join(compressed_signal))
    print(f'Ukuran File Setelah Kompresi: {compressed_size} bits')

    # Menghitung rasio kompresi
    compression_ratio = original_size / compressed_size
    print(f'Rasio Kompresi: {compression_ratio:.2f}')
```

Gambar 3.1 Program Utama
(Sumber: Dokumen Pribadi)

Saat program utama dijalankan, program akan membaca sinyal dan frekuensi dari audio yang akan dilakukan kompresi. Selanjutnya, sinyal audio akan dikompresi dengan fungsi compressor yang terdapat pada kelas Compressor yang akan dijelaskan pada bagian berikutnya. Secara umum, bagian ini bertanggung jawab atas proses kompresi, mengubah sinyal audio menjadi Kode Huffman, dan menyimpan hasilnya dalam variabel kode, yang berupa direktori. Fungsi kompresi ini juga mengembalikan hasil sinyal yang telah dikompresi.

Hasil kompresi memiliki manfaat signifikan, memungkinkan pengiriman audio yang lebih cepat dan efisien, serta mengoptimalkan ruang penyimpanan. Jika pengguna ingin mendengarkan kembali hasil audio yang telah dikompresi, dapat dilakukan dekompresi dengan fungsi decompressor yang ada pada kelas decompressor. Setelah memiliki hasil sinyal sebelum dan setelah dikompresi, program akan menampilkan perbandingan grafik sinyal sebelum dan sesudah dikompresi.

Program juga akan menampilkan ukuran dari file sebelum dan sesudah di kompresi. Setelah itu, program akan menghitung hasil rasio kompresi dengan rumus sebagai berikut:

$$compression\ ratio = \frac{original\ size}{compressed\ size} \quad (1)$$

Dengan *original size* didapat dengan mengalikan ukuran file audio asli dengan 8 untuk mengonversikan dalam bentuk bit. Sedangkan *compressed size* didapat dari panjang kode Huffman.

B. Hufmman Coding

```python
def __init__(self, orignal_signal):
    self.orignal_signal = orignal_signal

def frekuensi(self):
    freq_dict = {}
    for value in self.orignal_signal:
        if value not in freq_dict:
            freq_dict[value] = 0
        freq_dict[value] += 1
    return freq_dict

def frekuensi_inbuilt(self):
    return(dict(Counter(self.orignal_signal)))

def max_code_length(self, freq_dict):
    return(ceil(log(len(freq_dict), 2)))
```

Gambar 3.2. Menghitung Frekuensi Kemunculan
(Sumber: Dokumen Pribadi)

Sebelum melakukan kompresi dengan Kode Huffman, perlu untuk menyimpan frekuensi kemunculan amplitude pada titik-titik tertentu. Fungsi frekuensi akan menghitung banyak kemunculan masing-masing nilai dan mengembalikannya dalam bentuk direktori dengan nilai amplitudo sebagai *key* dan frekuensi kemunculan sebagai *value*. Fungsi-fungsi ini tersimpan pada kelas Compressor, yang nantinya hasilnya akan diolah untuk menyusun kode Huffman.

Fungsi-fungsi ini, menjadi dasar untuk pembentukan Kode Huffman. Frekuensi kemunculan amplitudo menjadi faktor kunci untuk menentukan representasi biner yang efisien untuk setiap nilai amplitudo pada saat proses kompresi dilakukan.

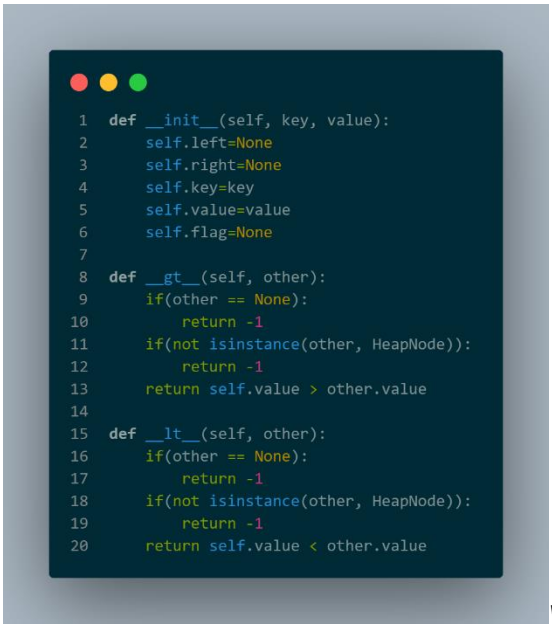
```python
def huffman_coding(self, freq_dict):
    freq_dict = {key: value for key, value in sorted(
        freq_dict.items(), key=lambda item: item[1])}
    heap = []

    for key in freq_dict:
        node = HeapNode(key, freq_dict[key])
        heapq.heappush(heap, node)

    cnt = 0
    while(len(heap) > 1):
        node1 = heapq.heappop(heap)
        node2 = heapq.heappop(heap)
        node1.flag = 0
        node2.flag = 1

        merged = HeapNode(None, node1.value + node2.value)
        merged.flag = (cnt % 2)
        cnt += 1

        merged.left, merged.right = node1, node2
        heapq.heappush(heap, merged)

    treePath = TreePath()
    treePath.paths(heap[0])

    temp_dict = treePath.temp_dict_var
    compressed_signal = []
    for elem in self.orignal_signal:
        compressed_signal.append(temp_dict[elem])

    code_dictVar = {}
    for key in temp_dict:
        code_dictVar[temp_dict[key]] = key

    return compressed_signal, code_dictVar

def maincompressor(self):
    freq_dict = self.frekuensi()
    compressed_signal, code_dictVar = self.huffman_coding(
        freq_dict)

    return compressed_signal, code_dictVar
```

Gambar 3.3. Proses Huffman Coding
(Sumber: Dokumen Pribadi)

Langkah awal pada proses *Huffman Coding* adalah mengurutkan frekuensi kemunculan elemen dari terkecil hinga terbesar pada direktori freq_dict. Selanjutnya, tiap elemen dan frekuensinya akan dibentuk node yang akan disimpan kedalam heap. Node ini digunakan untuk membangun pohon Huffman.

Selanjutnya akan dilakukan *looping* untuk menggabung dua node terkecil dari heap dan node hasilnya akan dimasukkan kembali ke dalam heap. Ini terus berlanjut hingga hanya ada satu node di heap, yang merupakan akar dari pohon Huffman.

Objek TreePath digunakan untuk menghasilkan *path* (kode) untuk setiap elemen dalam pohon Huffman. temp_dict berisi mapping antara elemen dan kode Huffman-nya. Objek ini tersimpan pada kelas TreePath. Setelah mendapatkan mapping Huffman code, sinyal asli dikompresi dengan mengganti setiap elemennya dengan kode Huffman yang sesuai. Program juga membuat kamus yang berisi mapping antara kode Huffman dan elemen (nilai amplitude) yang sesuai. Kamus ini bermanfaat untuk melakukan proses dekompresi nantinya.



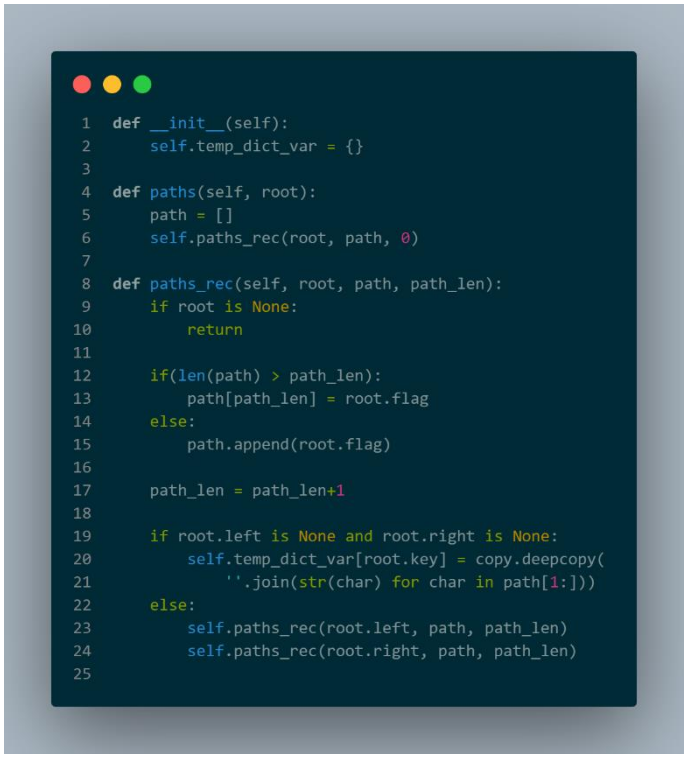
```
def __init__(self, key, value):
    self.left=None
    self.right=None
    self.key=key
    self.value=value
    self.flag=None

def __gt__(self, other):
    if(other == None):
        return -1
    if(not isinstance(other, HeapNode)):
        return -1
    return self.value > other.value

def __lt__(self, other):
    if(other == None):
        return -1
    if(not isinstance(other, HeapNode)):
        return -1
    return self.value < other.value
```

\

Gambar 3.4. Kelas HeapNode
(Sumber: Dokumen Pribadi)

HeapNode adalah kelas yang digunakan untuk merepresentasikan node dalam pohon biner yang digunakan dalam implementasi *Huffman coding*. Setiap node memiliki beberapa atribut penting, seperti, left dan right yang merupakan pointer ke node anak kiri dan kanan, key yang merupakan kunci atau elemen yang diwakili oleh node, dan value yang merupakan nilai atau frekuensi dari elemen yang diwakili oleh node.

Fungsi gt dan lt digunakan unruk mengurutkan node berdasarkan frekuensi kemunculannya. Node dengan nilai atau frekuensi yang lebih kecil menjadi prioritas yang lebih tinggi. Ini digunakan dalam pembangunan pohon Huffman, di mana node dengan frekuensi yang lebih rendah menjadi prioritas.



```
def __init__(self):
    self.temp_dict_var = {}

def paths(self, root):
    path = []
    self.paths_rec(root, path, 0)

def paths_rec(self, root, path, path_len):
    if root is None:
        return

    if(len(path) > path_len):
        path[path_len] = root.flag
    else:
        path.append(root.flag)

    path_len = path_len+1

    if root.left is None and root.right is None:
        self.temp_dict_var[root.key] = copy.deepcopy(
            ''.join(str(char) for char in path[1:]))
    else:
        self.paths_rec(root.left, path, path_len)
        self.paths_rec(root.right, path, path_len)
```

Gambar 3.5. Kelas TreePath
(Sumber: Dokumen Pribadi)

Kelas TreePath ini digunakan untuk membangun pohon Huffman dari node yang telah dibuat sebelumnya. Pembentukan ini dibangun dengan cara rekursif. Root menandakan node yang sedang diproses dan path adalah list yang digunakan untuk menyimpan path (kode Huffman) untuk elemen saat itu.

Jika root bukan None, program akan memeriksa apakah panjang *path* lebih panjang dari variable path len. Jika bernilai benar, berarti terdapat elemen dalam path saat ini, dan akan mengganti dengan nilai root.flag (0 atau 1) di posisi path_len. Jika tidak, root.flag akan ditambahkan ke path

Jika root adalah *leaf node* (tidak memiliki anak kiri dan kanan), akan dilakukan mapping dari elemen ke path (kode Huffman) ke dalam self.temp_dict_var. Jika root bukan merupakan *leaf node*, dilakukan rekursif pada anak kiri dan anak kanan untuk membangun path pohon.

C. Dekompresi File

```
class Decompressor:
    def __init__(self, compressed_signal, code_dict):
        self.compressed_signal = compressed_signal
        self.code_dict = code_dict

    def decompressor(self):
        orginal_signal = []
        for value in self.compressed_signal:
            orginal_signal.append(self.code_dict[value])

        return original_signal
```

Gambar 3.6. Kelas Dekompresi
(Sumber: Dokumen Pribadi)

Kelas Decompressor digunakan untuk mendekompresi sinyal yang telah dikompresi menggunakan algoritma *Huffman coding*. Metode Decompressor menggunakan kamus code_dict untuk mengonversi setiap kode Huffman dalam compressed_signal kembali menjadi elemen-elemen pada audio asli. Fungsi ini akan mengembalikan sinyal yang telah dikompresi menjadi bentuk audio yang bisa didengar.

IV. EKSPERIMEN

A. Kontruksi Kode Huffman

Penelitian ini menggunakan audio starwars yang bisa diakses pada *starwars soundtrack*. Audio ini memiliki format WAV yang selanjutnya akan diolah dan dilakukan kompresi dengan kode Huffman. Untuk membuat kode Huffman akan dilakukan perhitungan frekuensi kemunculan nilai amplitudo pada titik tertentu Tabel 4.1 menampilkan sebagian hasil perhitungan frekuensi kemunculan nilai amplitudo. Informasi lebih lengkap dapat diakses melalui *Amplitude Frequency*.

A	*freq*	*A*	*freq*	*A*	*freq*	*A*	*freq*	*A*	*freq*
-58	8	-50	12	-11	38	16	56	50	19
-55	12	-49	11	-9	32	20	49	47	24
-57	11	-45	8	-10	41	21	54	48	28
-59	14	-43	15	-7	40	3	43	54	19
-56	11	-41	9	-8	54	-4	42	51	15
-60	14	-40	10	-6	45	-5	40	52	17
-61	13	-46	10	-2	52	-12	31	53	12
-63	12	-42	9	-3	49	-16	11	55	12
-65	17	-39	5	-1	55	24	50	61	12
-62	18	-38	11	1	41	28	25	62	9
-69	31	-35	7	2	43	29	33	65	9
-66	25	-32	11	5	40	27	48	60	10
-64	20	-31	12	7	49	30	41	59	12
-68	22	-30	9	4	52	34	41	63	8
-70	16	-33	8	6	53	32	27	56	11
-71	9	-28	8	8	62	31	35	57	9
-73	13	-27	7	9	47	33	25	-19	7
-67	15	-24	11	11	43	35	24	-163	7
-72	14	-29	11	15	62	36	31	-230	5
-76	6	-22	8	19	52	37	28	-286	9
-77	5	-23	10	18	58	40	37	-337	7
-74	9	-26	7	17	50	41	19	-383	10
-75	16	-25	10	13	46	39	23	-428	5
-79	12	-21	11	10	42	38	23	-462	8
-54	5	-18	18	14	63	42	28	-494	7
-52	8	-20	11	12	54	45	20	-516	8
-48	9	-17	4	22	43	46	17	-533	6
-53	11	-13	24	23	51	43	26	-545	12
-51	11	-15	13	26	38	44	22	-550	9

Tabel 4.1 Frekuensi Kemunculan Amplitudo

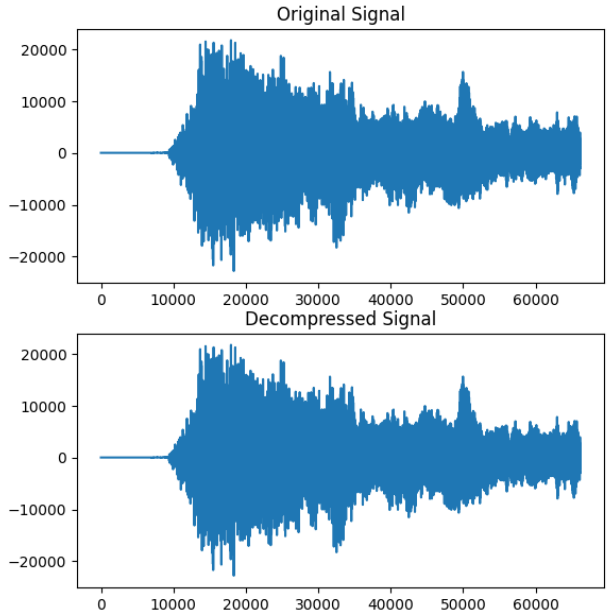
Setelah didapat hasil frekuensi untuk kemunculan amplitudo akan dibangun pohon Huffman dengan mengimplementasikan program *python* yang telah dibuat sebelumnya. Pohon Huffman yang terbentuk kemudian digunakan untuk menghasilkan kode Huffman. Setiap jalur dari akar pohon ke simpul-simpul daun akan menghasilkan representasi biner unik untuk setiap nilai amplitudo. Tabel 4.2 menunjukkan kode Huffman yang dihasilkan untuk beberapa nilai amplitudo. Untuk kode Huffman yang lebih lengkap dapat diakses melalui *Huffman Code*.

Kode	*A*	*Kode*	*A*	*Kode*	*A*	*Kode*	*A*
0	-983	111101	2179	11011011	-1507	10101111	-5516
10	-2533	1111101	2076	11011100	1269	10110000	-8015
110	2530	1111111	2177	10000001	-1598	10110001	-6270
111	3802	1000001	-1563	10000011	565	10110010	5502
1111	-13	1000011	-888	10000100	2715	10110011	6369
10000	-2975	1000101	-3657	10000101	3273	10110111	-594
10001	-3725	1000111	1011	10000110	-4122	10111000	-4356
10010	-7907	1001011	311	10000111	3975	10111001	-2059
10011	-6081	1001101	-1357	10001011	-637	10111010	-6387
10101	1857	1001111	-274	10001101	-4184	10111011	-4203
10111	-3390	1010001	2896	10001111	-263	10111100	-3342
11001	-2705	1010011	2615	10010001	188	10111101	-3817
11011	-1337	1010101	-3914	10010010	2858	10111110	-3370
11101	-1577	1010111	-3062	10010011	2508	10111111	7615
11110	-6670	1011011	-1759	10010101	2478	11000000	392
11111	-6155	1011101	1456	10010111	1495	11000001	2847
100001	367	1011110	6071	10011001	3271	11000011	-2775
100011	-300	1011111	-2136	10011011	-2289	11000101	1344
100101	-2814	1100001	2579	10011101	-2799	11000110	1249
100111	3451	1100011	412	10011110	-4579	11000111	-3380
101001	-2812	1100101	3660	10011111	-4501	11001001	2214
101011	-2748	1100111	2731	10100000	2998	11001011	1876
101101	-4279	1101000	2069	10100001	4742	11001100	-4636
101111	-2653	1101001	-1896	10100011	-2071	11001101	-4350
110000	3471	1101010	-4246	10100100	-5904	11001111	-314
110001	2870	1101011	4010	10100101	-1343	11010000	-4037
110011	2900	1101101	-3959	10100110	-4840	11010001	-4957
110101	-1188	1101111	-477	10100111	4397	11010010	-5692
110110	83	1110001	962	10101001	-2054	11010011	-4144
110111	-2224	1110011	-4216	10101010	-460	11010100	-4027
111000	-6831	1110101	2974	10101011	-5586	11010101	-5704
111001	-3637	1110111	3191	10101100	-4146	11010110	1607
111010	-3676	1111001	-1983	10101101	-5360	11010111	6170
111011	-3759	1111011	1496	10101110	6482	11011000	1905
111110	-1923	11011001	1831	11011101	7306	11011111	4724
111111	-2118	11011010	-2697	11011110	566	11100001	1094

Tabel 4.2. Representasi Kode Huffman

B. Proses Dekompresi

Setelah mendapatkan Kode Huffman untuk masing-masing nilai amplitudo, langkah selanjutnya adalah menyusun kode Huffman untuk seluruh sinyal audio. Hasil proses ini kemudian didekompresi dalam bentuk audio agar dapat didengarkan kembali. Gambar 4.1 menunjukkan perbandingan visual antara sinyal audio sebelum dan setelah dilakukan kompresi.



Gambar 4.1. Perbandingan Sinyal Audio
(Sumber: Dokumen Pribadi)

Pada gambar 4.1. terlihat bahwa tidak terjadi perubahan pada sinyal audio baik sebelum maupun setelah dilakukan kompresi. Struktur gelombang audio tetap terjaga, menandakan bahwa kualitas audio tidak mengalami degradasi meskipun terjadi kompresi. Perhatikan garis-garis kurva pada grafik yang merepresentasikan sinyal sebelum dan setelah kompresi, tidak terjadi perubahan.

Hasil ini mendukung bahwa teknik kompresi dengan Kode Huffman mampu mengurangi ukuran file audio tanpa mengorbankan kualitas audio. Meskipun jumlah memori yang digunakan lebih sedikit, perbandingan visual menunjukkan bahwa sinyal audio tetap identik. Oleh karena itu, teknik ini cocok digunakan dalam situasi saat terdapat batasan ukuran dalam pengiriman audio, seperti pada aplikasi *chatting*.

C. Perhitungan Rasio dan Kecepatan

Menggunakan formula (1) rasio kompresi pada sinyal audio didapat sebagai berikut:

Ukuran Sebelum Kompresi	1058752 bits
Ukuran Setelah Kompresi	1033676 bits

Tabel 4.3 Ukuran File Audio

Dari data diatas didapat rasio perbandingan ukuran sebelum dan setelah kompresi sebesar 1.02. Hal ini menunjukkan bahwa terdapat penurunan ukuran memori setelah dilakukan kompresi dengan kode Huffman. Penurunan ukuran memori yang terjadi membawa manfaat signifikan, terutama dalam hal penyimpanan dan waktu pengiriman. Kecepatan waktu pengiriman dapat dihitung dengan rumus sebagai berikut:

$$Kecepatan\ Pengiriman = \frac{ukuran\ memori}{waktu\ pengiriman} \quad (2)$$

Dari hasil perhitungan, jika waktu pengiriman audio selama 5 detik didapat kecepatan pengiriman sebagai berikut:

Kecepatan Sebelum	105875.2 bits / second
Kecepatan Sesudah	103367.6 bits / second

Tabel 4.4 Perbandingan Kecpatan Pengiriman

D. Pengaruh Terhadap Pengiriman Audio dalam Aplikasi Chatting

Dari hasil eksperimen didapat bahwa ukuran *file* setelah dikompresi terjadi penurunan. Ukuran *file* yang lebih kecil ini berpengaruh pada kecepatan pengiriman audio. Pada tabel 4.4, dapat dilihat bahwa audio yang sudah dikompresi lebih cepat dalam melakukan pengiriman. Tentu hal ini dapat menghemat waktu dan menjadi lebih efisien.

Beberapa aplikasi *chat* memberi batasan dalam melakukan pengiriman audio. Sehingga, terkadang saat hendak melakukan pengiriman *file* berukuran besar, terdapat kendala yang dialami, seperti, *file* audio yang terpotong, waktu pengiriman yang lama, atau penurunan kualitas audio saat dikirimkan. Belum lagi masalah memori yang harus disediakan. Oleh karena itu, dengan kompresi Huffman jumlah memori yang digunakan menjadi lebih sedikit dan juga berdasarkan gambar 4.1 tidak terjadi perubahan kualitas audio yang dikirim.

Meskipun teknik kompresi Huffman membuktikan efektivitasnya dalam penurunan ukuran file audio dan peningkatan kecepatan pengiriman dalam beberapa situasi, perlu diakui bahwa tidak satu metode pun dapat menjadi solusi universal. Keberhasilan teknik kompresi Huffman sangat tergantung pada karakteristik dan sifat data audio yang dihadapi.

Pada kondisi tertentu, seperti saat menghadapi jenis audio dengan pola nilai amplitudo yang unik, metode kompresi Huffman mungkin tidak memberikan pengurangan ukuran file yang signifikan. Sebaliknya, situasi ini dapat mengakibatkan overhead yang tidak perlu pada proses kompresi, bahkan mungkin tidak sebanding dengan keuntungan yang diperoleh.

Dalam konteks ini, penting untuk selalu mempertimbangkan karakteristik khusus dari data audio yang dihadapi. Beberapa situasi mungkin memerlukan pendekatan kompresi yang berbeda atau teknik khusus yang dirancang untuk menangani pola-pola tertentu dalam data audio.

V. KESIMPULAN

Kode Huffman telah terbukti sebagai alternatif yang efektif dalam proses kompresi audio. Eksperimen menunjukkan bahwa setelah dilakukan kompresi, ukuran memori audio menjadi lebih kecil yang menghasilkan kecepatan pengiriman audio yang lebih cepat dengan tetap menjaga kualitas audio asli. Oleh karena itu, metode ini cocok untuk diaplikasikan dalam melakukan pengiriman audio di aplikasi *chatting* karena efektif dan efisien.

Meskipun demikian, penting untuk mempertimbangkan bahwa pada beberapa kondisi khusus, seperti pola nilai amplitudo yang

unik pada jenis audio tertentu, metode kompresi Huffman mungkin tidak memberikan pengurangan ukuran *file* yang signifikan. Walaupun demikian, teknik ini tetap dapat dianggap sebagai pilihan yang layak untuk kompresi audio, memberikan alternatif yang efisien dan efektif melakukan kompresi sebuah audio.

VI. Ucapan Terima Kasih

Puji Syukur terhadap Allah SWT karena berkat Rahmat dan karunia-Nya, makalah berjudul "Optimasi Memori dalam Pengiriman Audio pada Aplikasi Chat dengan Kode Huffman" dapat terselesaikan dengan baik dan tepat waktu. Terima kasih kepada orang tua dan teman-teman seperjuangan yang selalu memberi semangat dalam proses pengerjaan. Terima kasih juga kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. selaku dosen mata kuliah Matematika Diskrit atas bimbingannya dan Bapak Dr. Ir. Rinaldi Munir, MT yang sudah menyediakan laman untuk berbagi proses pembelajaran selama ini.

Daftar Pustaka

[1] T. Hidayat, "Comparison of Lossless Compression Schemes for WAV Audio Data 16-Bit between," 2019.

[2] Bedrus dan Quiros, "Comparison of Huffman Algorithm and Lempel-Ziv Algorithm for audio, image and text compression," 2016.

[3] Hidayat, "A Critical Assessment of Advanced Coding Standards for Lossless Audio Compression," pp. 1-10, 1948.

[4] R. Munir, "Matematika Diskrit Revisi Ketujuh," 2020.

[5] B. A. Setyo, "Optimasi Kapasitas Steganografi Digital dengan Teknik Kompresi Huffman Coding," 2022.

[6] MasterClass, "A Guide to Audio File Formats," 3 November 2021. [Online]. Available: https://www.masterclass.com/articles/a-guide-to-audio-file-formats. [Diakses 2023 December 2023].

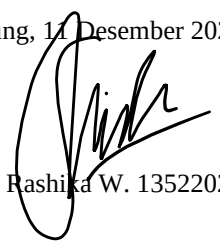
[7] Rendra, "Berapa Batas Mengirim File Via Whatsapp," AnakTekno, 7 December 2023. [Online]. Available: https://anaktekno.id/aplikasi/whatsapp/berapa-batas-mengirim-file-via-whatsapp-25162. [Diakses 8 December 2023].

[8] N. S. Jayant dan E. Y. Chen, "Audio compression: Technology and applications," *AT&T Technical Journal*, vol. 74, pp. 23-34, 1995.

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, 11 Desember 2023



Debrina Veisha Rashika W. 13522025