Komparasi Algoritma Huffman Code dengan Run Length Encoding pada Kompresi Teks dalam PDF

Elizabeth Hanov, 23031554055
Program Studi S1 Sains Data
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Surabaya
elizabeth.23055@mhs.unesa.ac.id

Aghnia Alya Amarilla, 23031554102
Program Studi S1 Sains Data
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Surabaya
aghnia.23102@mhs.unesa.ac.id

Nashita Erha Fitri, 23031554116
Program Studi S1 Sains Data
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Surabaya
nashita.23116@mhs.unesa.ac.id

***Abstract*— Proyek ini membahas kompresi file teks yang terdiri dari kumpulan karakter atau string yang membentuk satu kesatuan. Dengan meningkatnya kebutuhan akan efisiensi penyimpanan dan kecepatan transfer data, proyek ini membandingkan efektivitas dua algoritma, yaitu Run Length Encoding dan Huffman Code, dalam mengurangi ukuran file teks dalam PDF. Melalui analisis yang dilakukan, proyek ini bertujuan untuk mengidentifikasi kelebihan dan kekurangan masing-masing algoritma dalam konteks kompresi data. Hasil dari penelitian ini diharapkan dapat memberikan wawasan yang lebih baik mengenai teknik kompresi yang dapat diterapkan untuk meningkatkan efisiensi pengelolaan data.**

***Keywords*— Kompresi, File Teks, Algoritma Huffman Code, Algoritma Run Length Encoding**

I. Pendahuluan

Dalam perkembangan teknologi saat ini, komputer digunakan untuk membantu dan mempercepat kinerja manusia. Salah satu tantangan yang dihadapi dalam komunikasi data adalah ukuran file yang seringkali sangat besar, yang dapat memperlambat waktu pengiriman dan memerlukan ruang penyimpanan yang signifikan. Hal ini semakin menjadi relevan dalam konteks file PDF, yang sering digunakan untuk menyimpan dokumen teks, gambar dan grafik. File PDF dapat memiliki ukuran yang besar, terutama ketika berisi elemen-elemen multimedia atau banyak halaman.

Untuk mengatasi masalah ini, teknik kompresi data menjadi sangat penting terutama untuk file PDF yang berisi teks sesuai dengan topik yang kami angkat. Kompresi teks bertujuan untuk mengurangi redundansi dalam data, sehingga ukuran file dapat diperkecil tanpa kehilangan informasi penting. Dalam hal ini, algoritma kompresi yang efisien sangat penting untuk memastikan bahwa dokumen tetap dapat diakses dan dibagikan dengan mudah.

Dua algoritma kompresi yang populer adalah Huffman dan Run Length Encoding (RLE). Algoritma Huffman memiliki tiga tahapan untuk mengkompres data, yaitu pembentukan pohon, encoding dan decoding. Berbeda dengan algoritma Huffman yang bekerja berdasarkan karakter per karakter, tetapi algoritma Run Length Encoding bekerja sederetan karakter yang berurutan secara berulang.

Dalam proyek ini, akan dianalisis perbandingan efektivitas algoritma kompresi Huffman dan Run Length Encoding (RLE) dalam mengurangi ukuran file PDF tanpa kehilangan informasi penting. Selain itu, proyek ini juga akan membahas tantangan yang dihadapi dalam menjaga kualitas data setelah proses kompresi dan dekompresi pada file PDF. Pengaruh kompleksitas waktu dan penggunaan sumber daya terhadap kinerja algoritma kompresi akan dievaluasi, serta perbedaan rasio kompresi yang dihasilkan oleh algoritma Huffman dan RLE ketika diterapkan pada berbagai jenis konten dalam file PDF, seperti teks. Dengan demikian, diharapkan proyek ini dapat memberikan wawasan yang lebih mendalam mengenai efektivitas dan efisiensi algoritma kompresi dalam konteks file PDF.

II. Dasar Teori

A. Kompresi

Kompresi adalah proses pengkodean informasi menggunakan bit atau informasi-bearing unit yang lain yang lebih rendah dari pada representasi data yang tidak terkodekan dengan suatu sistem encoding tertentu. Proses kompresi merupakan proses mereduksi suatu data untuk menghasilkan representasi digital yang padat atau mampat (Compact) namun tetap dapat mewakili kuantitas informasi yang terkandung pada data tersebut. Dalam mekanisme kompresi, data yang telah dikompresi harus melalui proses dekompresi menggunakan metode yang sama agar dapat dikembalikan ke bentuk aslinya dengan benar dan dapat diakses seperti semula. Kompresi data terdiri dari dua jenis teknik, yaitu teknik lossless dan teknik lossy.

1. *Teknik Lossless*

   Teknik kompresi dimana data yang akan dikompres adalah data yang dapat dikembalikan ke bentuk aslinya setelah proses kompresi, sehingga dapat disusun ulang dan dipulihkan seperti semula. Teknik ini cocok untuk mengkompresi data berupa teks sebab tidak menghilangkan data sama sekali, sehingga dalam konteks topik yang kami angkat, tergolong ke dalam teknik lossless.

2. *Teknik Lossy*

Teknik lossy merupakan teknik kompresi data yang menghapus sebagian data asli untuk mengurangi ukurannya. Pada teknik ini, beberapa informasi yang dianggap kurang penting atau tidak terlalu berpengaruh terhadap kualitas data akan dihilangkan, sehingga ukuran file dapat dikurangi. Meskipun demikian, setelah dekompresi, data yang dipulihkan tidak akan persis sama dengan data asli. Teknik ini kerap tergolong lebih efektif untuk mengkompresi data gambar, audio, atau video.

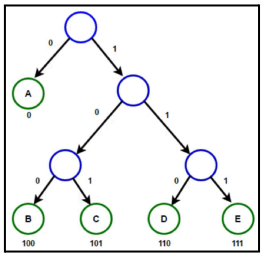
B. *Algoritma Huffman Code*

Algoritma Huffman merupakan metode di mana tidak ada data yang hilang selama proses pemampatan, sehingga informasi tetap utuh dan dapat disimpan dalam bentuk aslinya. Secara umum, pada algoritma huffman karakter-karakter yang digunakan bentuk ASCII akan diubah kedalam bentuk bit-bit yakni 0 dan 1 mirip dengan sistem sandi Morse, di mana ia menciptakan kode untuk setiap karakter. karakter-karakter ini dikelompokkan berdasarkan frekuensi munculnya, sehingga karakter-karakter yang paling sering muncul ditempatkan pada pohon Huffman dengan jarak kode biner yang lebih pendek, dan karakter-karakter yang lebih jarang muncul ditempatkan pada pohon Huffman dengan jarak kode biner yang lebih panjang.

Pohon Huffman terdiri dari simpul (node) yang terhubung oleh garis, yang mewakili jarak kode biner. Pada akhirnya, setiap karakter akan diwakili oleh kode biner yang unik, sehingga karakter dapat direpresentasikan dalam bentuk kode biner. Kode yang dihasilkan memiliki panjang bit yang lebih pendek dibandingkan dengan representasi aslinya. Proses kompresi menggunakan algoritma Huffman melibatkan tiga langkah utama:

1. Membangun pohon yang terdiri dari simpul-simpul yang mewakili karakter beserta frekuensi kemunculannya.
2. Melakukan encoding di mana setiap karakter diberikan identitas dalam bentuk bilangan biner untuk disimpan.
3. Melakukan decoding yang merupakan proses kebalikan dari encoding, di mana bilangan biner yang lebih pendek diubah kembali menjadi karakter asli tanpa kehilangan data.

Berikut adalah contoh struktur pohon Huffman. Dari struktur tersebut, dapat diketahui bahwa:



- Karakter A dikodekan biner menjadi 00
- Karakter B dikodekan biner menjadi 100
- Karakter C dikodekan biner menjadi 101
- Karakter D dikodekan biner menjadi 110
- Karakter E dikodekan biner menjadi 111

C. *Algoritma Run Length Encoding*

Algoritma ini efektif untuk mengompres data yang memiliki karakter yang berulang dan serupa, dengan cara menyimpan informasi tersebut sebagai satu entitas tunggal alih-alih menyimpan setiap karakter secara terpisah, RLE bekerja dengan mengurangi ukuran data melalui pengulangan string karakter, di mana string yang berulang disebut RUN dan dikodekan dalam dua bit. Bit pertama menunjukkan jumlah pengulangan, sedangkan bit kedua menunjukkan karakter yang diulang. Metode ini juga mencakup teknik seperti Repetition Suppression, yang menggantikan angka atau huruf yang berulang dengan satu huruf yang mewakili jumlahnya, dan Pattern Substitution, yang mengganti kata-kata dengan huruf atau simbol tertentu. Sebagai contoh; diketahui karakter BBBBBBBBBAAAAANGGMMM yang memiliki 20 karakter. Ketika dikompresi : B = 9 merepresentasikan karakter B muncul sebanyak 9 kali, A = 4 merepresentasikan karakter A muncul sebanyak 4 kali, N = 1 merepresentasikan karakter N muncul sebanyak 1 kali, G = 2 merepresentasikan karakter G muncul sebanyak 2 kali, dan M = 3 merepresentasikan karakter M muncul sebanyak 3 kali. Maka didapat, hasil kompresinya menjadi 9B4A1N2G3M yang memiliki 10 karakter.

III. Implementasi

A. *Implementasi Algoritma Huffman Code*

Algoritma Huffman merupakan metode kompresi data yang efektif dengan menghasilkan kode biner unik berdasarkan frekuensi kemunculan setiap simbol dalam data. Prosesnya diawali dengan menghitung frekuensi tiap simbol, yang disimpan dalam sebuah struktur data seperti dictionary. Frekuensi ini kemudian digunakan untuk membangun pohon Huffman, di mana setiap simbol direpresentasikan sebagai sebuah node dengan bobot yang mencerminkan frekuensinya. Dua node dengan bobot terkecil secara berulang diambil dari antrian prioritas, digabungkan menjadi node baru dengan bobot yang merupakan jumlah dari kedua node tersebut, lalu dimasukkan kembali ke dalam antrian. Proses ini berlanjut hingga hanya tersisa satu node, yaitu akar pohon Huffman. Setelah pohon terbentuk, ia ditelusuri secara rekursif untuk menghasilkan kode biner unik bagi setiap simbol. Simbol yang lebih sering muncul diberikan kode lebih pendek, sementara simbol yang jarang muncul mendapatkan kode lebih panjang.

Proses encoding dilakukan dengan mengganti setiap simbol dalam data asli menggunakan kode biner yang dihasilkan, menghasilkan string biner sebagai hasil akhirnya. Sementara itu, proses decoding dilakukan dengan menelusuri pohon Huffman berdasarkan setiap bit dalam string biner untuk mengembalikannya ke data asli. Kompleksitas waktu algoritma ini meliputi $O(n)$ untuk menghitung frekuensi simbol, $O(k \log k)$ untuk membangun pohon, dan $O(m)$ untuk encoding dan decoding. Total kompleksitasnya adalah $O(n + k \log k)$, dengan n adalah panjang data, k jumlah simbol unik, dan m panjang string biner yang dihasilkan. Dari sisi ruang, algoritma ini membutuhkan $O(k)$ untuk menyimpan pohon Huffman dan $O(m)$ untuk menyimpan hasil encoding.

Algoritma Huffman unggul dalam mengkompresi data dengan distribusi simbol yang tidak merata, di mana simbol yang sering muncul diberi kode lebih pendek sehingga efisiensi meningkat. Namun, metode ini kurang optimal untuk data dengan distribusi simbol yang seragam dan dapat menimbulkan overhead pada dataset yang kecil. Meski

demikian, sifat optimalnya menjadikan Huffman pilihan yang sangat baik untuk kompresi teks dengan distribusi simbol yang tidak merata. Berikut merupakan pseudocode dari algoritma Huffman Code yang kami gunakan.

```
procedure Huffman_Code(input data: string, output encoded_data: string, decoded_data: string, encode_runtime: float, decode_runtime: float) { Melakukan Huffman Encoding dan Decoding pada string input "data" } { Masukan: data adalah string yang akan di-encode dan di-decode } { Luaran: encoded_data adalah string hasil encoding, decoded_data adalah string hasil decoding, encode_runtime dan decode_runtime adalah waktu eksekusi masing-masing }

Deklarasi: symbol_with_probs: dictionary nodes: list of Node huffman_tree: Node encoded_data, decoded_data: string start_time, stop_time: float current_node: Node decoded_output: list of string

Algoritma:

{Langkah 1: Encoding}

start_time ← waktu_saat_ini()

symbol_with_probs ← Calculate_Probability(data)

nodes ← buat daftar Node untuk setiap simbol dan probabilitas dalam symbol_with_probs

while (panjang(nodes) > 1) do

        nodes ← urutkan nodes berdasarkan prob

        left ← hapus elemen pertama dari nodes

        right ← hapus elemen pertama dari nodes

        left.code ← 0

        right.code ← 1

        newNode ← Node(left.prob + right.prob, left.symbol + right.symbol, left, right)

        tambahkan newNode ke nodes

endwhile

huffman_tree ← nodes[0]

huffman_encoding ← Calculate_Codes(huffman_tree)

encoded_data ← Output_Encoded(data, huffman_encoding)

stop_time ← waktu_saat_ini()

encode_runtime ← stop_time - start_time

{Langkah 2: Decoding}

start_time ← waktu_saat_ini()

current_node ← huffman_tree

decoded_output ← []

for setiap bit dalam encoded_data do

        if (bit = '1') then

        current_node ← current_node.right

        else

        current_node ← current_node.left

        endif

        if (current_node.left = null and current_node.right = null) then

        tambahkan current_node.symbol ke decoded_output

        current_node ← huffman_tree

        endif

endfor

decoded_data ← gabungkan semua elemen decoded_output menjadi string

stop_time ← waktu_saat_ini()

decode_runtime ← stop_time - start_time

return encoded_data, decoded_data, encode_runtime, decode_runtime

end procedure
```

B. Implementasi Algoritma Run Length Encoding

Algoritma Run Length Encoding (RLE) adalah metode kompresi data yang dirancang khusus untuk menangani data dengan pola elemen yang berulang secara berturut-turut. Konsep utamanya adalah menggantikan rangkaian simbol yang sama dengan pasangan nilai yang terdiri dari jumlah

kemunculan simbol tersebut dan simbol itu sendiri. Cara kerjanya dimulai dengan membaca data masukan secara berurutan sambil menghitung jumlah simbol yang muncul berturut-turut. Ketika menemukan rangkaian simbol yang sama, algoritma mencatat jumlahnya dan mengganti rangkaian tersebut dengan pasangan (jumlah, simbol). Sebagai contoh, jika data masukan adalah "AAAABBBCCDAA", maka hasil kompresinya menjadi "4A3B2C1D2A".

Proses encoding dilakukan dengan mengganti setiap rangkaian simbol berulang dalam data asli menggunakan pasangan nilai yang sesuai, sehingga menghasilkan representasi yang lebih ringkas. Sebaliknya, decoding dilakukan dengan membaca pasangan nilai dari data terkompresi dan merekonstruksi data aslinya dengan mengulang simbol sesuai jumlah yang tercatat dalam setiap pasangan.

Dari segi kompleksitas waktu, algoritma RLE memiliki efisiensi *O(n)*, karena setiap elemen data hanya diproses sekali untuk menghitung urutan berulang. Untuk kompleksitas ruang, algoritma ini membutuhkan *O(k)*, di mana *k* adalah jumlah rangkaian simbol berulang yang berbeda. Dalam skenario terbaik, ketika data terdiri dari banyak urutan berulang yang panjang, algoritma menghasilkan kompresi yang sangat efisien. Namun, jika data memiliki sedikit pola berulang atau karakter yang sangat bervariasi, hasil kompresi dapat menjadi kurang optimal, bahkan lebih besar daripada data aslinya. Berikut merupakan pseudocode dari algoritma Run Length Encoding yang kami gunakan.

```
procedure RLE(input data: string, output encoded_data: string, decoded_data: string, encode_runtime: float, decode_runtime: float) { Melakukan Run-Length Encoding (RLE) dan Decoding pada string input "data" } { Masukan: data adalah string yang akan di-encode dan di-decode } { Luaran: encoded_data adalah string hasil encoding, decoded_data adalah string hasil decoding, encode_runtime dan decode_runtime adalah waktu eksekusi masing-masing }

Deklarasi: encoding, decoding: string i, count: integer start_time, stop_time: float

Algoritma:

{Langkah 1: Encoding RLE}
start_time ← waktu_saat_ini()
encoding ← ""
i ← 0

while (i < panjang(data)) do
        count ← 1
        while (i + 1 < panjang(data) and data[i] = data[i + 1]) do
        count ← count + 1
        i ← i + 1
        endwhile
        encoding ← encoding + data[i] + string(count)
        i ← i + 1
endwhile

stop_time ← waktu_saat_ini()
encode_runtime ← stop_time - start_time

{Langkah 2: Decoding RLE}
start_time ← waktu_saat_ini()
decoding ← ""
i ← 0

while (i < panjang(encoding)) do
        char ← encoding[i]
        count ← integer(encoding[i + 1])
        decoding ← decoding + char * count
        i ← i + 2
endwhile

stop_time ← waktu_saat_ini()
decode_runtime ← stop_time - start_time

return encoding, decoding, encode_runtime, decode_runtime

end procedure
```

C. *Perbandingan Kompleksitas Algoritma 1 dan Algoritma 2*

Huffman Code dan Run-Length Encoding (RLE) menawarkan pendekatan kompresi data yang berbeda, baik dalam cara kerja maupun kompleksitasnya. Huffman Code mengandalkan distribusi simbol untuk menghasilkan kode variabel yang optimal, terutama pada data dengan frekuensi simbol yang tidak merata. Kompleksitas waktu algoritma ini adalah *O(n + k log k)*, yang mencerminkan proses membangun pohon Huffman melalui pengurutan simbol berdasarkan frekuensinya. Algoritma ini juga membutuhkan ruang tambahan untuk menyimpan pohon dan hasil encoding, sehingga lebih intensif dalam hal penggunaan sumber daya.

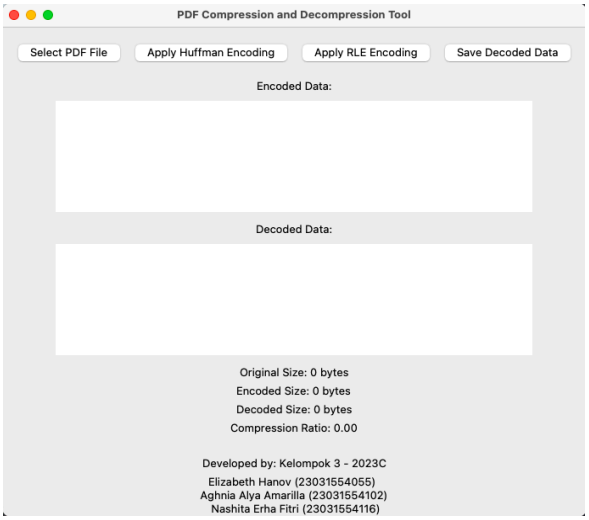
Sementara itu, RLE lebih sederhana dan efisien untuk data dengan urutan simbol yang berulang panjang. Kompleksitas waktunya *O(n)*, karena hanya membutuhkan satu kali pemindaian untuk mengganti urutan simbol yang sama dengan pasangan jumlah dan simbolnya. Namun, efisiensinya sangat tergantung pada pola data—untuk data yang tidak memiliki banyak pengulangan, hasil kompresinya bisa kurang optimal atau bahkan lebih besar daripada data aslinya.

Perbedaan utama antara kedua algoritma ini terletak pada fleksibilitas dan aplikasinya. Huffman Code lebih unggul dalam menangani berbagai jenis data dengan distribusi simbol yang bervariasi, meskipun memerlukan lebih banyak komputasi. Di sisi lain, RLE lebih hemat sumber daya tapi hanya efektif untuk data dengan banyak elemen berulang. Oleh karena itu, pemilihan algoritma yang tepat harus disesuaikan dengan sifat data yang akan dikompresi dan kebutuhan akan efisiensi komputasi.

IV. HASIL DAN DISKUSI

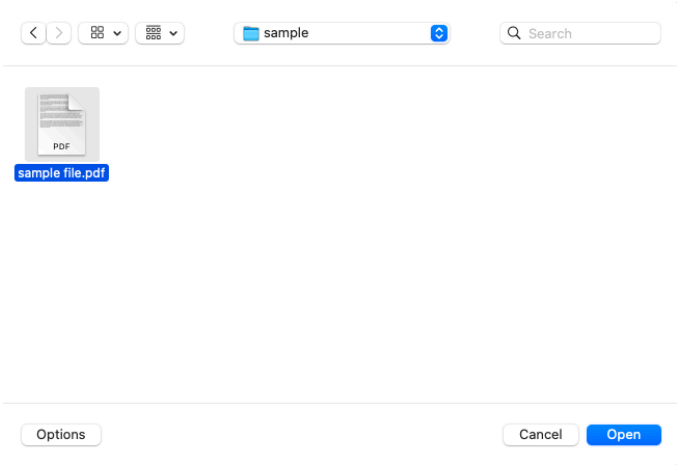
Untuk mempermudah meninjau proses kompresi teks dari file PDF menggunakan algoritma **Huffman Code** dan **Run Length Encoding (RLE)**, kami telah membuat antarmuka grafis (GUI) yang sederhana dan mudah digunakan.

1. Tampilan awal



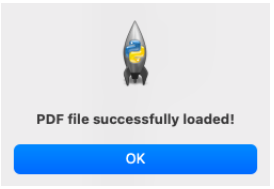
Pada tampilan awal ini, pengguna akan diminta untuk memilih file PDF menggunakan tombol *Select PDF File*. Setelah file berhasil dimuat, pengguna dapat memilih salah satu algoritma kompresi, yaitu *Apply Huffman Encoding* untuk Huffman Code atau *Apply RLE Encoding* untuk Run Length Encoding.

2. Tampilan memilih file PDF yang akan diproses



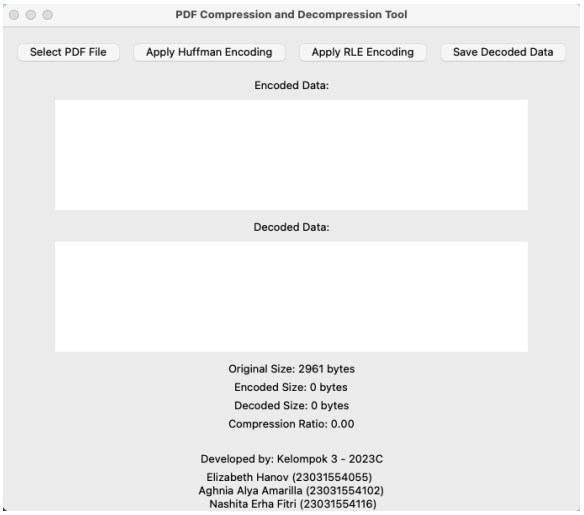
Berikut merupakan tampilan setelah pengguna menekan tombol *Select PDF File*. Pengguna dapat menavigasi folder, memilih file PDF yang diinginkan, dan menekan tombol *Open* untuk memuat file.

3. Tampilan saat sudah berhasil input file



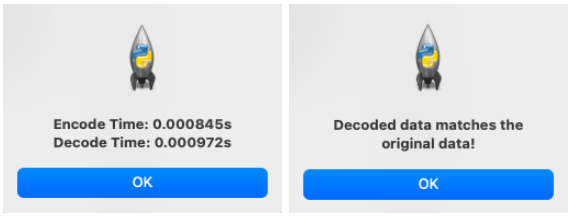
Saat sudah berhasil input file PDF, akan muncul message box seperti ini.

4. Tampilan saat File PDF sudah berhasil dimuat



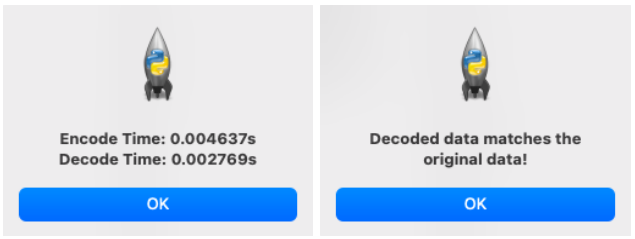
Saat file sudah berhasil dimuat, original size akan berubah sesuai dengan ukuran file yang dimuat.

5. Tampilan ketika memilih Algoritma Huffman Encoding



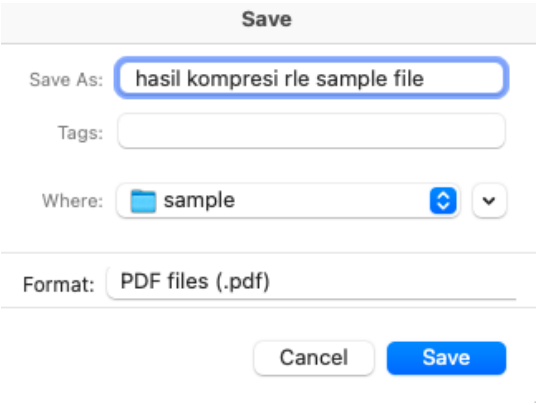
Pada tampilan tersebut, kita menggunakan algoritma Huffman Coding. Setelah berhasil menekan tombol *Apply Huffman Encoding*, data dikompresi menjadi bitstream yang ditampilkan pada bagian Encoded Data. Proses encoding berlangsung sangat cepat, dengan waktu 0.000845 detik, dan data hasil kompresi dapat didekompres kembali menjadi teks asli melalui tombol Save Decoded Data, dengan waktu decoding 0.000972 detik. Validasi keberhasilan terlihat dari pesan "Decoded data matches the original data!" yang menunjukkan data asli berhasil dipulihkan tanpa kehilangan informasi. Ukuran data asli adalah 2961 bytes, sedangkan hasil encoding berukuran 9477 bytes, dan hasil decoding berukuran 2257 bytes, dengan rasio kompresi sebesar 3.20.

6. Tampilan ketika memilih Algoritma Run Length Encoding

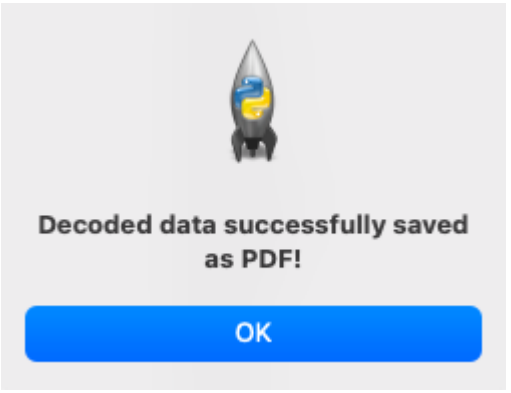


Sebaliknya, pada tampilan ini kita menggunakan algoritma Run Length Encoding (RLE). Setelah berhasil menekan tombol Apply RLE Encoding, data dikompresi menjadi pola karakter berulang yang ditampilkan pada bagian Encoded Data. Proses encoding berlangsung sangat cepat, yaitu 0.004637 detik, sementara proses decoding membutuhkan waktu 0.002769 detik, dan hasil dekompresi kembali ke bentuk asli yang valid, seperti yang dikonfirmasi oleh pesan "Decoded data matches the original data!". Ukuran data asli adalah 2961 bytes, sementara hasil encoding menjadi 4476 bytes, dan hasil decoding berukuran 2257 bytes, dengan rasio kompresi sebesar 1.51, yang menunjukkan efisiensi kompresi dari metode RLE.

7. Tampilan untuk Menyimpan File Hasil Kompresi



Ketika sudah berhasil menekan tombol *Save Decoded Data,* pengguna akan diminta memberi nama file hasil kompresi untuk disimpan.



Ketika file telah berhasil disimpan, akan muncul message box seperti pada gambar diatas.

Setelah menjalankan kedua algoritma kompresi menggunakan antarmuka grafis (GUI), berikut ini adalah tabel hasil kompresi dan dekompresi menggunakan dua algoritma pengkodean, yaitu Huffman Code dan Run Length Encoding (RLE). Kedua algoritma ini menunjukkan kinerja yang berbeda dalam hal rasio kompresi dan waktu pemrosesan. Hasil yang diperoleh memberikan gambaran mengenai efektivitas masing-masing algoritma dalam menangani data dengan karakteristik yang berbeda.

TABLE I. TABEL HASIL KOMPRESI DAN DEKOMPRESI ALGORITMA HUFFMAN CODE

Nama File	Algoritma Huffman Code					
	Original Size	*Encoded Size*	*Decoded Size*	*Rasio Kompresi*	*Encode Time*	*Decode Time*
sample file.pdf	2961 bytes	9477 bytes	2257 bytes	3.20	0.0008 45 s	0.0009 72 s

Algoritma Huffman Code efektif dalam mengurangi ukuran data dengan rasio kompresi **3.20**, yang berarti data terkompresi menjadi sekitar sepertiga ukuran aslinya. Algoritma ini menggunakan pengkodean berdasarkan frekuensi karakter, dengan karakter sering diberi kode lebih pendek. Proses encoding dan decoding cepat, masing-masing **0.000845 detik** dan **0.000972 detik**, berkat pohon biner yang

efisien. Namun, ada overhead pada encoding karena penyimpanan metadata seperti tabel frekuensi dan pohon, yang menyebabkan ukuran hasil kompresi lebih besar (**9477 bytes**). Meski demikian, dekompresi berhasil mengembalikan data ke ukuran asli (**2257 bytes**) tanpa kehilangan informasi.

TABLE II. TABEL HASIL KOMPRESI DAN DEKOMPRESI ALGORITMA RUN LENGTH ENCODING (RLE)

Nama File	Algoritma Run Length Encoding					
	Original Size	*Encoded Size*	*Decoded Size*	*Rasio Kompresi*	*Encode Time*	*Decode Time*
sample file.pdf	2961 bytes	4476 bytes	2257 bytes	1.51	0.0046 37 s	0.0027 69 s

Algoritma Run Length Encoding (RLE) menghasilkan rasio kompresi **1.51**, menunjukkan ukuran data hasil encoding sedikit lebih besar daripada data asli karena pola berulang yang tidak cukup signifikan. Waktu pemrosesan encoding dan decoding adalah **0.004637 detik** dan **0.002769 detik**, lebih lambat dibandingkan Huffman Code. Meskipun sederhana, RLE tetap bersifat lossless, mengembalikan data ke ukuran asli (**2257 bytes**) tanpa kehilangan informasi. RLE efektif untuk data dengan pola berulang yang jelas, tetapi kurang efisien untuk data acak atau beragam.

Berdasarkan hasil pada tabel sebelumnya, berikut ini adalah tabel komparasi yang membandingkan kinerja Huffman Code dan Run Length Encoding (RLE) berdasarkan ukuran file dan rasio kompresi. Tabel ini menunjukkan perbedaan dalam efektivitas masing-masing algoritma dalam mengurangi ukuran file dan rasio kompresi yang dihasilkan.

TABLE III. TABEL KOMPARASI HASIL KOMPRESI ALGORITMA 1 DAN ALGORITMA 2

Algoritma	Ukuran File		*Rasio Kompresi*
	Sebelum Kompresi	*Setelah Kompresi*	
Huffman Code	2961 bytes	9477 bytes	3.20
Run Length Encoding (RLE)	2961 bytes	4476 bytes	1.51

Secara keseluruhan, algoritma Huffman Code menunjukkan keunggulan yang signifikan dalam hal rasio kompresi dibandingkan dengan algoritma Run-Length Encoding (RLE). Pada data uji, algoritma Huffman Code menghasilkan rasio kompresi sebesar **3.20**, jauh lebih tinggi dibandingkan dengan algoritma RLE yang hanya mencapai rasio kompresi **1.51**. Hal ini menunjukkan bahwa Huffman Code lebih efisien dalam mengkompresi data yang memiliki distribusi frekuensi karakter yang tidak merata. Meskipun ukuran file setelah kompresi menggunakan algoritma Huffman Code sedikit lebih besar dibandingkan dengan RLE, rasio kompresinya tetap lebih tinggi, yang berarti lebih banyak informasi yang dikompresi dalam ukuran yang lebih kecil.

Di sisi lain, algoritma RLE lebih cocok diterapkan pada data yang memiliki pola berulang yang jelas dan sederhana. Pada kasus data uji ini, algoritma RLE tidak memberikan hasil yang optimal karena pola berulang tidak cukup dominan. Meskipun demikian, algoritma RLE tetap merupakan algoritma yang sederhana dan mudah diimplementasikan, serta dapat lebih cepat diterapkan pada data yang sesuai.

Berdasarkan hasil percobaan kami, dapat disimpulkan bahwa algoritma Huffman Code lebih efektif untuk kompresi data dengan distribusi karakter yang tidak merata, karena memberikan rasio kompresi yang lebih tinggi. Sementara itu, algoritma RLE lebih tepat digunakan pada data dengan pola berulang yang jelas. Pemilihan algoritma kompresi yang tepat sangat bergantung pada karakteristik data yang akan dikompresi.

FILE PYTHON AT DRIVE

Proyek UAS DAA Kelompok 3 2023C

VIDEO LINK AT YOUTUBE

https://youtu.be/q69FAmtX2jI

ACKNOWLEDGMENT

Penyusunan laporan dan penyelesaian proyek ini tidak lepas dari dukungan dan masukan berbagai pihak. Dengan tulus, kami mengucapkan terima kasih kepada Ibu Fadhilah Qalbi Annisa, S.T., M.Sc. dan Ibu Kartika Chandra Dewi, S.Si., M.Si. selaku dosen mata kuliah *Desain dan Analisis Algoritma*, atas bimbingan dan arahan yang diberikan selama proses pengerjaan proyek ini. Tidak lupa, kami juga mengucapkan terima kasih kepada rekan-rekan yang telah memberikan masukan dan dukungan moral, sehingga proyek ini dapat diselesaikan dengan baik.

REFERENSI

[1] Pujianto, Mujito, Basuki Hari Prasetyo, dan Danang Prabowo, "Perbandingan Metode Huffman dan Run Length Encoding Pada Kompresi Document," *InfoTekJar: Jurnal Nasional Informatika dan Teknologi Jaringan*, vol. 5, no. 1, pp. 216-223, September 2020.

[2] Sylvia Rahma dan Aditya Pranapanca, "Analisis Kompresi dan Dekompresi Data Teks dan Audio dengan Algoritma Run Length Encoding (RLE)," JINACS: Journal of Informatics and Computer Science, vol. 02, no. 04, pp. 313-320, November 2021.