KOMIK (Konferensi Nasional Teknologi Informasi dan Komputer)
Volume 6, Nomor 1, November 2022
DOI: 10.30865/komik.v6i1.5739
ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

Analisis Perbandingan Kinerja Algoritma Huffman Dan Algoritma Levenstein Dalam Kompresi File Dokumen Format .RTF

Dwi Asdini*, Dito Putro Utomo

Fakultas Ilmu Komputer dan Teknologi Informasi, Program Studi Teknik Informatika, Universitas Budi Darma, Medan, Indonesia
Email: 1*dwiasdini@gmail.com
Email Penulis Korespondensi: dwiasdini@gmail.com

Abstrak-File dokumen dengan format .RTF merupakan file yang cukup sering digunakan namun memiliki ukuran yang cukup besar. Ukuran file yang besar sering mengalami kendala pada saat mengirim file dimana proses pengiriman dapat memakan waktu yang lama dan dapat menghabiskan banyak ruang penyimpanan karena terlalu banyak data berukuran besar yang tersimpan. Hal ini tentu sangat menyulitkan jika ada banyak file yang harus disimpan, sementara media penyimpanan memiliki batasan ukuran. Untuk mengatasi masalah ini maka perlu dilakukan kompresi untuk memampatkan data. Untuk memilih algoritma yang optimal dalam mengkompresi file dokumen maka perlu dilakukan perbandingan antara algoritma Huffman dan algoritma Levenstein dengan mengukur kinerja berdasarkan Ratio Of Compression (RC), Compression Ratio (CR), Redundancy (Rd) dan Space Saving (SS). Setelah dilakukan perhitungan maka didapat hasil algoritma yang paling optimal dalam mengkompresi file dokumen yaitu algoritma Huffman dengan space saving yang lebih tinggi dan ukuran hasil kompresi lebih rendah dibandingkan dengan algoritma Levenstein. Hal ini dikarenakan semakin besar space saving (ss) maka kompresinya semakin bagus dan semakin cepat.

Kata Kunci: Kompresi; Perbandingan; File Dokumen; Huffman; Levenstein

Abstract-Document files with the .RTF format are files that are quite often used but have a fairly large size. Large file sizes often experience problems when sending files where the sending process can take a long time and can consume a lot of storage space because too much large data is stored. This is certainly very difficult if there are many files that must be stored, while the storage media has a size limit. To overcome this problem it is necessary to do compression to compress the data. To choose the optimal algorithm for compressing document files, it is necessary to compare the Huffman algorithm and the Levenstein algorithm by measuring performance based on the Ratio Of Compression (RC), Compression Ratio (CR), Redundancy (Rd) and Space Saving (SS). After the calculation, the results of the most optimal algorithm in compressing document files are obtained, namely the Huffman algorithm with higher space saving and lower compression results compared to the Levenstein algorithm. This is because the greater the space saving (ss) the compression is getting better and faster.

Keywords: Compression; Comparison; Document File; Huffman; Levenstein

1. PENDAHULUAN

Meningkatnya penggunaan data digital saat ini mengakibatkan banyaknya data yang tersimpan di dalam media penyimpanan. Sehingga sering terjadi kendala dalam mentransmisi data, misalnya pada saat mengirim data dapat memakan waktu yang cukup lama ataupun kurangnya kapasitas memori karena terlalu banyak data berukuran besar yang tersimpan. Hal ini tentu sangat menyulitkan jika ada banyak *file* yang harus disimpan, sementara media penyimpanan memiliki batasan ukuran. Untuk mengatasi hal ini maka ukuran data harus lebih kecil agar dapat menyimpan lebih banyak *file* di media penyimpanan dengan melakukan pemampatan data. Pemampatan data dapat dilakukan pada berbagai jenis *file*, seperti *file* dokumen sebagai *file* yang paling banyak disimpan didalam media penyimpanan.

File dokumen merupakan file yang berisi informasi yang berasal dari dokumen Microsoft Word, Microsoft Excel, Microsoft Access, dan beberapa dokumen lain yang terdiri dari beberapa karakter, angka, tanda baca, maupun gambar. Sebagian besar penggunaan memori digital pada media penyimpanan berupa file dokumen. File dokumen yang memiliki format file dengan ukuran besar yang terus tersimpan akan membuat tempat penyimpanan menjadi penuh, belum lagi ditambah dengan ukuran dari jenis file lainnya. Salah satu format file dokumen dengan ukuran besar yaitu file dokumen format .RTF atau Rich Text Format. Ukuran file yang relatif besar menjadi masalah dalam penyimpanan. Dimana file dengan ukuran besar cenderung lebih membutuhkan banyak ruang penyimpanan dan mempengaruhi waktu transmisi data. Oleh karena itu, dibutuhkan adanya proses pemampatan data[1].

Pemampatan data atau kompresi data merupakan proses memadatkan data dimana ukuran data yang besar diubah menjadi ukuran yang lebih kecil. Kompresi data sangan penting karena mengurangi kebutuhan penyimpanan data, meningkatkan kecepatan transfer data, dan mengurangi kebutuhan bandwidth. Kompresi data dibagi menjadi dua yaitu kompresi lossy dan kompresi lossless. Kompresi lossy memampatkan data dengan menghilangkan sejumlah informasi sehingga terjadi perubahan data. Sedangkan kompresi lossless memampatkan dengan tetap menjaga data, dan memiliki derajat kompresi yang lebih rendah. Kompresi dapat dilakukan dengan menerapkan algoritma untuk menghasilkan ukuran file yang lebih kecil.

Ada banyak algoritma yang dapat digunakan untuk mengkompresi file dengan kelebihan dan kekurangan masing-masing. Untuk mengetahui kinerja suatu algoritma dalam mengkompresi file maka dapat dihitung parameter kinerjanya berdasarkan Ratio Of Compression (RC), Compression Ratio (CR), Redundancy (Rd) dan Space Saving (SS). Memilih algoritma yang sesuai sangat diperlukan karena tidak semua algoritma dapat mengkompresi file dengan baik. Algoritma yang cukup terkenal dalam melakukan kompresi yaitu algoritma Huffman dan algoritma Levenstein.

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

Algoritma Huffman dan algoritma Levenstein merupakan algoritma yang cukup populer dalam mengkompresi file dokumen. Pada dasarnya kedua algoritma ini memiliki prinsip yang sama yaitu memampatkan data, namun memiliki cara kerja yang berbeda. Algoritma Huffman, dibuat oleh seorang mahasiswa MIT bernama David Huffman pada tahun 1952, adalah salah satu metode kompresi teks tertua dan paling terkenal. Prinsip pengkodean algoritma Huffman mirip dengan kode morse, yaitu setiap karakter dikodekan dengan beberapa kode bit, dimana karakter yang sering muncul dikodekan dengan rangkaian bit yang lebih pendek dan karakter yang jarang muncul dikodekan dengan rangkaian bit yang lebih panjang[2]. Vladimir Levenstein mengembangkan algoritma Levenstein tahun 1968. Algoritma Levenstein merupakan algoritma yang prosesnya melalui tahapan-tahapan tertentu baik pada saat pengkodean maupun pembacaan sandi. Algoritma Levenstein bekerja dengan menghitung jumlah minimum transformasi satu string menjadi string lainnya, termasuk penghapusan, penyisipan, dan pertukaran[3].

Dikarenakan kedua algoritma ini paling banyak digunakan untuk mengkompresi file maka dilakukan perbandingan untuk mengetahui algoritma mana dari kedua algoritma ini yang menghasilkan ukuran file yang lebih kecil setelah dilakukan kompresi agar dapat digunakan untuk membantu menghemat ruang penyimpanan.

Pada penelitian terdahulu yang ditulis oleh Helbert Sinaga, Poltak Sihombing, dan Handrizal pada tahun 2018 tentang "Perbandingan Algoritma Huffman dan Run Length Encoding Untuk Kompresi File Audio" menyatakan bahwa implementasi Algoritma Huffman dan Run Length Encoding memiliki tujuan untuk mengkompresi file sehingga ukuran file yang dikompresi lebih kecil dari file aslinya. Hasil penelitian menunjukkan algoritma Huffman lebih baik dalam mengkompresi file *.mp3 maupun *.wav. Dimana rasio kompresi pada file audio *.mp3 dengan Algoritma Huffman rata-rata 1.204% untuk RLE -94.44% dan kompresi file audio *.wav rata-rata 28.954% untuk RLE adalah 5.91%[4].

Berdasarkan penelitian terdahulu oleh Novita Sari dan Khairul Ummi pada tahun 2020 tentang "Perancangan Aplikasi Kamus Bahasa Minang Indonesia dan Indonesia Minang Menggunakan Algoritma Levenshtein" menyatakan bahwa Aplikasi ini menerapkan metode Levenstein, sebuah algoritma yang banyak digunakan untuk berbagai bidang, misalnya mesin untuk pencarian browser, untuk pemeriksaan ejaan atau biasa disebut checker, dikte, untuk pengenalan suara, analisis DNA, dll. Dalam aplikasi ini, algoritma Levenstein telah diterapkan untuk melakukan pencarian kata[5].

Berdasarkan penelitian yang dilakukan oleh Supiyandi dan Okta Frida pada tahun 2018 tentang "Analisis Perbandingan Pemampatan Data Teks Dengan Menggunakan Metode Huffman Dan Half-Byte" menyatakan bahwa algoritma Huffman memiliki ratio lebih tinggi sehingga pemampatan yang dilakukan lebih baik daripada algoritma Half-Byte[6].

Berdasarkan penelitian terdahulu yang dilakukan oleh Dian Pratiwi dan Taronisokhi Zebua pada tahun 2019 tentang "Analisi Perbandingan Kinerja Algoritma Fixed Length Binary Endcoding Dan Algoritma Elias Gamma Code Dalam Kompresi File Teks" menyatakan bahwa hasil uji kompresi dengan kedua metode menunjukkan bahwa file yang berukuran besar berhasil dikompresi mejadi ukuran lebih kecil dibandingkan sebelum dikompresi dengan perbandingan kinerja algoritma Fixed Length Binary Encoding lebih baik dalam mengkompresi file jenis .RTF[7].

Berdasarkan penelitian terdahulu yang dilakukan oleh Akhmad Pahdi pada tahun 2018 tentang "Algoritma Huffman dalam Pemampatan dan Enkripsi Data" menyatakan bahwa algoritma Huffman merupakan algoritma yang digunakan untuk mengkompresi dan mengenkripsi file, hasil ukuran file dari proses ini menjadi lebih kecil dan langsung dienkripsi dalam susunan karakter baru dan tidak dapat terbaca dengan mudah[8].

2. METODOLOGI PENELITIAN

2.1 Metodologi Penelitian

Metodologi Penelitian merupakan rangkaian tahapan yang dilakukan dalam penyelesaian masalah yang dibahas pada sebuah penelitian. Metodologi Penelitian ini dibuat agar memudahkan dalam penyelesaian sebuah penelitian. Adapun metodologi penelitian penelitian terdapat pada gambar 1 sebagai berikut:

Gambar 1. Metodologi Penelitian

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

2.2 Kompresi

2.2.1 Kompresi Data

Kompresi data adalah proses mengubah sekumpulan data ke dalam bentuk kode untuk memperkecil ukuran file sebelum disimpan atau dipindahkan ke media penyimpanan. Hal ini berguna untuk menghemat ruang penyimpanan yang digunakan dan dapat mempersingkat waktu pengiriman data. Terdapat dua proses utama dalam kompresi yaitu kompresi dan dekompresi atau pengembalian data. Tujuan dari kompresi yaitu mengurangi jumlah bit yang digunakan untuk menyimpan atau mengirim informasi[9]. Pada dasarnya kompresi data terbagi menjadi dua kelompok besar yaitu:

a. Data kompresi *lossy*
   Dalam kompresi data lossy, proses pemampatan data dengan mengurangi bahkan menghilangkan informasi sehingga menyebabkan perubahan data dibandingkan dengan sebelum dilakukan kompresi. Jika terjadi kesalahan selama kompresi, selama itu tidak mengubah sampel data, kesalahan itu masih diperbolehkan dan biasa digunakan dalam kompresi. Kompresi data lossy efektif bila diterapkan pada gambar, film, dan audio digital.
b. Data Kompresi *lossless*
   Rasio kompresi lebih rendah, akurasi data sebelum dan sesudah kompresi tetap dipertahankan.Lossless mengkompresi data tanpa menghilangkan data aslinya. Kesalahan kompresi data lossless tidak dapat ditoleransi, sehingga data pra-kompresi harus sama persis dengan data yang dikompresi. Kompresi data lossless juga dikenal sebagai kompresi reversible karena data asli dapat dipulihkan sepenuhnya dengan dekompresi[2].
   Berikut parameter yang biasa digunakan untuk menganalisis kinerja kompresi[10], yaitu:
   1. *Ratio of Compression* (RC)
      Ratio of Compression adalah nilai yang membandingkan ukuran bit data sebelum dikompresi dengan ukuran bit data yang dikompresi. Secara matematis dapat ditulis sebagai berikut:

      $$\boldsymbol{R_C = \frac{Ukuran\ bit\ sebelum\ dikompresi}{Ukuran\ bit\ setelah\ dikompresi}} \quad (1)$$

   2. *Compression Ratio* (CR)
      Compression Ratio adalah persentase antara bit data terkompresi dan bit data tidak terkompresi.
      $C_R = \frac{Ukuran\ bit\ setelah\ dikompresi}{Ukuran\ bit\ sebelum\ dikompresi}$ (2)
   3. *Space Saving* (SS)
      Space Saving adalah jumlah penyimpanan yang dihemat oleh kompresi, persentase antara data yang tidak terkompresi dan data terkompresi.
      $S_s = (1 - CR)x100\%$ (3)
   4. *Radudancy* (Rd)
      Radudancy (Rd) adalah selisih persentase yang dihasilkan antara ukuran data sebelum dikompresi dan setelah dikompresi.
      $R_d = 100\% - CR$ (4)

2.2.2 *File* Dokumen Format .RTF

File dokumen merupakan kumpulan data yang berisi informasi dalam bentuk teks. Data dari dokumen yang berkaitan dengan huruf, angka, simbol, dan teks lain yang digunakan dalam perhitungan, nama, dan alamat dalam database adalah contoh entri data dokumentual dimana mencakup karakter, angka, dan tanda baca. Input dan output dari data dokumen diwakili oleh set karakter atau input kode yang dikenali oleh input komputer[11].

File dokumen yang digunakan pada penelitian ini berformat .RTF atau Rich Text Format. RTF dapat menyimpan sebagian besar pemformatan dokumen, berbagai font, dan mendukung objek dan gambar. File RTF dapat dibaca dan diedit pada program pengolah kata apapun, seperti Microsoft Word, OpenOffice.org, dan lainnya. RTF juga dapa dibuka pada system operasi apapun seperti Mac dan Linux.

2.3 Metode dan Tahapannya

2.3.1 Algoritma Huffman

Algoritma Huffman adalah algoritma yang dikembangkan oleh David A. Huffman tahun 1952. Algoritma Huffman menggunakan prinsip pengkodean yang sama dengan kode Morse, yaitu setiap karakter hanya dikodekan dengan string beberapa bit, di mana karakter yang sering terlihat dikodekan dengan string bit pendek dan karakter jarang terlihat dikodekan dengan string bit lebih panjang. Algoritma Huffman adalah algoritma kompresi lossless yang sangat cocok untuk mengkompresi file dokumen. Itu sebabnya algoritma ini banyak digunakan dalam program kompresi. Teknik kompresi algoritma ini yaitu dengan menggunakan kode berupa bit untuk mempresentasikan data karakter[2]. Cara kerja algoritma ini yaitu:

a. Hitung jumlah jenis karakter dan angka dari setiap karakter dalam file.

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

b. Urutkan setiap jenis karakter secara berurutan dari yang memiliki jumlah karakter paling sedikit hingga yang memiliki jumlah karakter paling banyak.
c. Buat pohon biner dalam urutan karakter terkecil hingga terbesar dan tetapkan kode untuk setiap karakter.
d. Ganti data berdasarkan pohon biner dengan kode bit.
e. Menyimpan jumlah bit untuk kode bit terbesar, tipe karakter diurutkan dari frekuensi keluaran tertinggi hingga terkecil dan data diubah menjadi kode bit sebagai data terkompresi[2].

2.3.2 Algoritma Levenstein

Algoritma Levenstein adalah kode universal untuk bilangan bulat non-negatif yang dikembangkan oleh Vladimir Levenstein tahun 1968. Algoritma Levenstein adalah sebuah algoritma dimana proses melewati tahapan tertentu baik pada saat enkripsi maupun pada saat deskripsi. Untuk proses enkripsi dalam algoritma Levenstein angka 0 adalah satu 0[12]. Untuk menulis kode positif n, lakukan langkah berikut:

a. Deklarasikan angka pertama C dengan 1. Inisialisasi kode saat ini pada string kosong.
b. Ambil nilai biner n tanpa awalan 1 dan tambahkan ke kode saat ini.
c. Nyatakan M sebagai jumlah bit yang ditambahkan pada langkah 2.
d. Jika M ≠0, tambahkan C dengan 1 dan ulangi langkah 2, tetapi dengan M bukan n.
e. Jika M = 0, tambahkan 1, diikuti 0 pada C ke kode saat ini dan berhenti[12].

Tabel berikut merupakan nilai dari kode Levenstein. Dalam tabel, setiap bagian dari kode Levenstein dipisahkan oleh spasi dimaksudkan untuk memudahkan indikasi bagian dari kode tersebut.

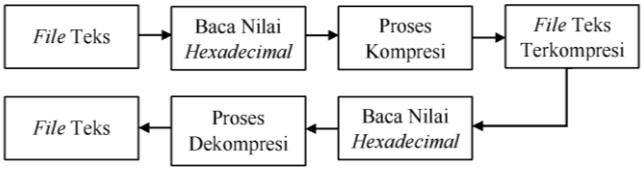
Tabel 1. Kode Levenstein

N	Kode Levenstein
0	0
1	10
2	110 0
3	110 1
4	1110 0 00
5	1110 0 01
6	1110 0 10
7	1110 0 11
8	1110 1 000
9	1110 1 001
10	1110 1 010
11	1110 1 011
12	1110 1 100
13	1110 1 101
14	1110 1 110
15	1110 1 111
16	11110 0 00 0000
17	11110 0 00 0001

3. HASIL DAN PEMBAHASAN

Sebelum melakukan proses perancangan, maka dibutuhkan beberapa analisis masalah. Analisis dapat memberikan gambaran yang komprehensif tentang masalah yang dianalisis dengan mengidentifikasi dan mengevaluasi untuk memberikan solusi atas masalah yang dibahas. Tahapan yang dilakukan yaitu kompresi dan dekompresi.

Pada tahap awal Analisa yang dilakukan mencari file dokumen dengan format .RTF. yang selanjutnya diubah menjadi nilai hexadecimal dengan menggunakan aplikasi HxD. Setelah didapat nilai hexadecimal maka selanjutnya mengkompresi file dengan menerapkan algoritma Huffman dan algoritma Levenstein. Dibawah ini merupakan alur sederhana dalam proses kompresi file dokumen:



Gambar 2. Prosedur Kompresi dan Dekompresi *File* Dokumen

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

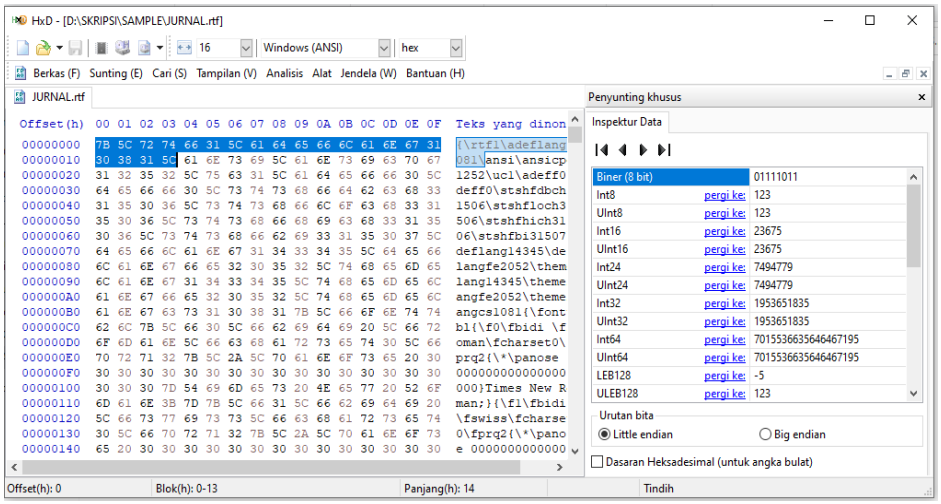
3.1 Penerapan Algoritma Huffman

Penelitian ini akan membahas dua algoritma, yang pertama yang akan dibahas yaitu algoritma Huffman. Algoritma Huffman adalah algoritma kompresi lossless yang sangat cocok untuk mengkompresi file dokumen. Sebelum melakukan tahap pertama, penulis mencari file dokumen format .RTF untuk dilakukan kompresi. Berikut adalah contoh file yang akan dikompresi:

Tabel 2. Sampel *File* Dokumen

No	Nama *File*	Ukuran Sebelum Dikompresi
1	Jurnal.RTF	8.20 MB
2	Skripsi.RTF	49.6 MB
3	TestData.RTF	1.00 MB

Berdasarkan contoh sampel diatas, maka file tersebut diubah menjadi nilai hexadecimal dengan menggunakan aplikasi HxD.



Gambar 3. Nilai *Hexadecimal* Sampel *File* Dokumen

Untuk melakukan kompresi penulis mengambil 20 nilai hexadecimal untuk mewakili file dokumen dengan format .RTF yang telah diubah dengan aplikasi HxD.

Tabel 3. Nilai *Hexadecimal* Yang Belum Dikompresi

7B	5C	72	74	66	31	5C	61	64	65
66	6C	61	6E	67	31	30	38	31	5C

Kemudian urutkan nilai hexadecimal berdasarkan nilai frekuensi yang terbesar hingga frekuensi terkecil. Terlihat pada tabel 4.2 dibawah ini ukuran string yang dihasilkan sebelum dilakukan kompresi yaitu 160 bit.

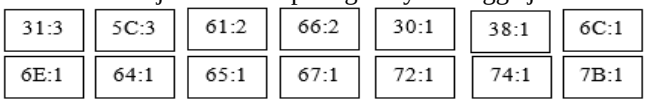
Tabel 4. Nilai *Hexadecimal* Yang Belum Dikompresi

No	Nilai Hexadecimal	Freq	Nilai Biner	Bit	Freq x Bit
1	31	3	110001	8	24
2	5C	3	1011100	8	24
3	61	2	1100001	8	16
4	66	2	1100110	8	16
5	30	1	110000	8	8
6	38	1	111000	8	8
7	6C	1	1101100	8	8
8	6E	1	1101110	8	8
9	64	1	1100100	8	8
10	65	1	1100101	8	8
11	67	1	1100111	8	8
12	72	1	1110010	8	8
13	74	1	1110100	8	8
14	7B	1	1111011	8	8
	Total	20	-	-	160

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

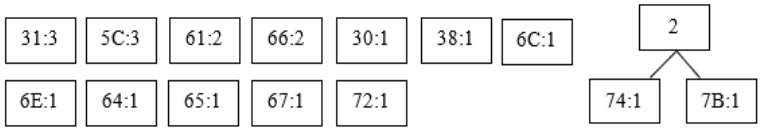
Selanjutnya untuk mengkompresi file dokumen dengan algoritma Huffman maka ikuti langkah-langkah yang telah dijelaskan sebelumnya dengan membentuk pohon biner dan setiap nilai hexadecimal akan memiliki identitas berupa bilangan biner.

a. Urutkan setiap nilai hexadecimal dari jumlah nilai paling banyak hingga jumlah nilai paling sedikit.



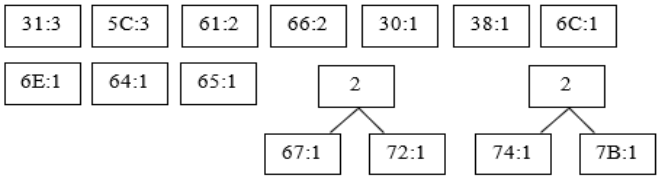
Gambar 4. Langkah 1

b. Akarkan nilai hexadecimal berdasarkan nilai frekuensi terkecil dimulai dari 74 dan 7B bernilai masing-masing 1.



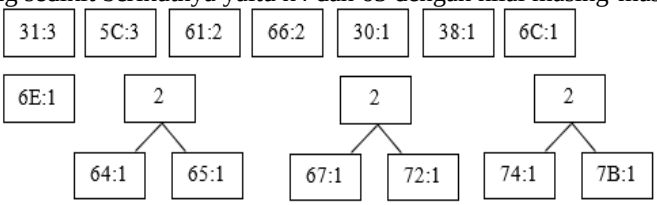
Gambar 5. Langkah 2

c. Lanjutkan ke nilai frekuensi paling sedikit setelahnya yaitu 67 dan 72 bernilai masing-masing 1.



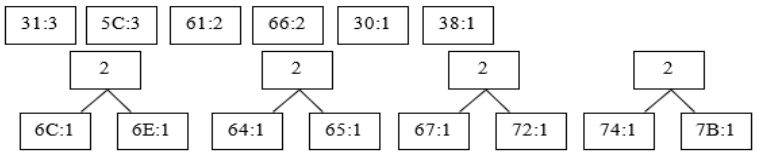
Gambar 6. Langkah 3

d. Lanjutkan ke nilai paling sedikit berikutnya yaitu k4 dan 65 dengan nilai masing-masing 1.



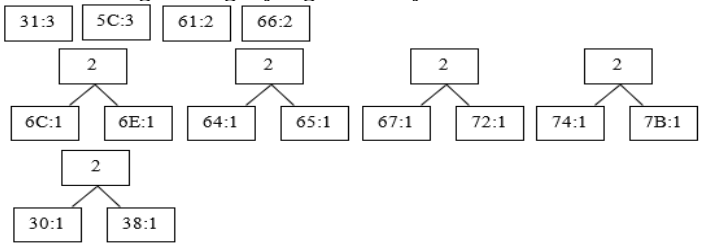
Gambar 7. Langkah 4

e. Akar selanjutnya yaitu 6C dan 6E dengan nilai masing-masing 1.



Gambar 8. Langkah 5

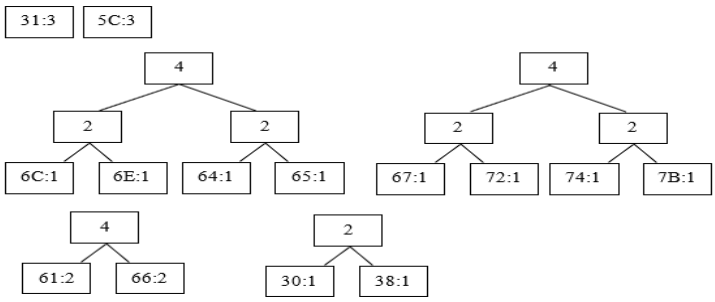
f. Lanjut ke akar dengan nilai masing-masing 1 yang terakhir yaitu 30 dan 38.



Gambar 9. Langkah 6

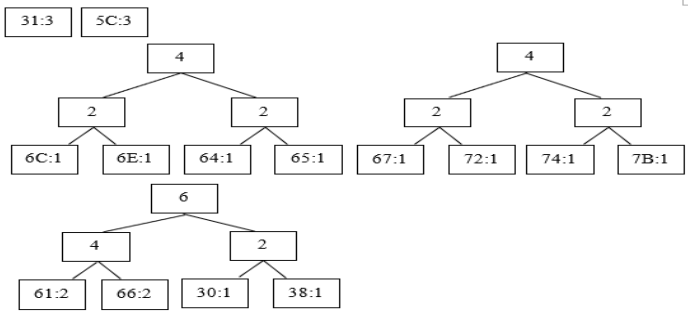
g. Setelah itu gabungkan akar dengan nilai paling kecil 67 dan 72 gabungkan dengan nilai 74 dan 7B karna memiliki nilai masing-masing 2. Nilai 6C dan 6E gabungkan dengan nilai 64 dan 65 yang memiliki nilai masing-masing 2. Lalu akarkan nilai paling sedikit berikutnya yaitu 61 dan 66 dengan nilai masing-masing adalah 2.

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99



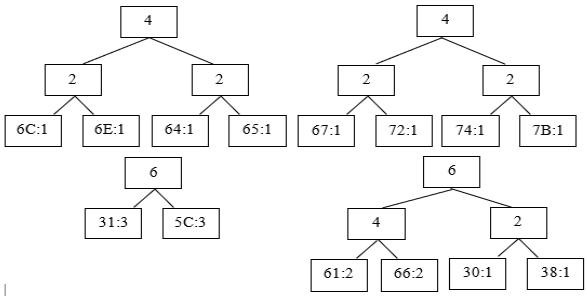
Gambar 10. Langkah 7

h. Gabungkan akar 30 dan 38 yang bernilai 2 dengan nilai 61 dan 66 bernilai 4.



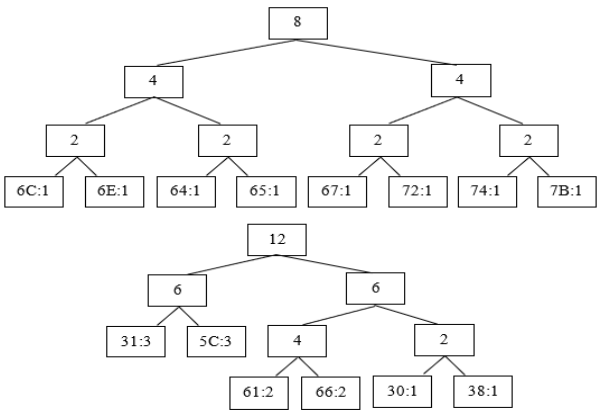
Gambar 11. Langkah 8

i. Akar selanjutnya yaitu 31 dan 5C dengan nilai masing-masing 3.



Gambar 12. Langkah 9

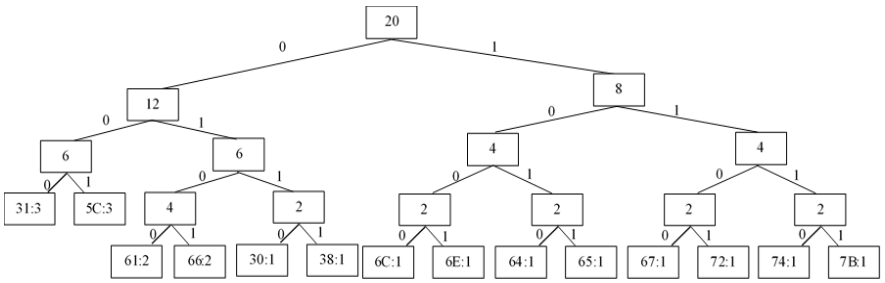
j. Gabungkan akar dengan jumlah nilai yang sama, akar nilai 4 dengan yang bernilai 4, akar yang bernilai 6 dengan akar yang bernilai 6.



Gambar 13. Langkah 10

k. Tahap terakhir yaitu gabungkan semua akar yang dibentuk menjadi satu dan didapatlah total nilai 20. Kemudian beri bit 0 pada cabang akar sebelah kiri dan bit 1 pada cabang akar sebelah kanan.

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99



Gambar 14. Hasil

Nilai bit yang dihasilkan dari pohon biner diatas dapat dilihat pada tabel 4 dibawah ini.

Tabel 5. Hasil Dari Pohon Biner

No	Nilai Hexadecimal	Freq	Codeword	Bit	Freq x Bit
1	31	3	0	3	9
2	5C	3	1	3	9
3	61	2	100	4	8
4	66	2	101	4	8
5	30	1	110	4	4
6	38	1	111	4	4
7	6C	1	1000	4	4
8	6E	1	1001	4	4
9	64	1	1010	4	4
10	65	1	1011	4	4
11	67	1	1100	4	4
12	72	1	1101	4	4
13	74	1	1110	4	4
14	7B	1	1111	4	4
	Total	20	-	-	74

Langkah selanjutnya yaitu menyusun kembali string bit yang dihasilkan dari proses kompresi.

Tabel 6. *String Bit* Hasil Kompresi

7B	**5C**	**72**	**74**	**66**	**31**	**5C**	**61**	**64**	**65**
1111	1	1101	1110	101	0	1	100	1010	1011
66	**6C**	**61**	**6E**	**67**	**31**	**30**	**38**	**31**	**5C**
101	1000	100	1001	1100	0	110	111	0	1

Berdasarkan tabel 5 string bit yang dihasilkan dari proses pengkompresian file dokumen menggunakan algoritma Huffman dapat dituliskan sebagai berikut:
"11110011101111001010000010100101010111010110000100100111000000110011100000 1"
Kemudian dari hasil total bit x frekuensi yang bernilai 74, maka perlu dilakukan penambahan jumlah bit karena 74 tidak habis dibagi 8. Dimana didalam komputer, suatu karakter direpresentasikan oleh bilangan ASCII sebanyak 8 bit dari bilangan biner. Apabila bit suatu data lebih pendek dari 8 bit maka biasanya tidak akan bisa dibaca oleh komputer sebelum dilakukannya proses encoding. Pada proses encoding, bit data tersebut di-encode setiap 8 bitnya sehingga membentuk suatu karakter yang dapat dibaca komputer. Berikut cara untuk mencari padding dan flagging.

a. Jika sisa bagi panjang string bit adalah 0 maka tambahkan 00000001 untuk dinyatakan sebagai bit akhir.
b. Jika sisa bagi panjang string bit adalah n (1, 2, 3, 4, 5, 6, 7) maka tambahkan 0 sebanyak 7 – n +"1" di akhir string bit. Lalu ditambahkan bilangan biner dari 9 – n untuk dinyatakan sebagai bit akhir.

Table 7. Penambahan *Padding* dan *Flagging*

Padding	*Flagging*
$74\ mod\ 8 = 2 = n$	$9 - n$
$n + 1$"=7 - 2 + "1" = 5 = 000001	$9 - 2 = 7 = 00000111$

Maka tambahkan 000001 dan 00000111 ke akhir string bit, menjadi:
"11110011101111001010000010100101010111010110000100100111000000110011100000 10000010000010000111".
Setelah ditambah padding dan flagging total bit menjadi 88 bit. Lakukan pemisahan dengan membagi bit menjadi per delapan bit. Maka langkah selanjutnya ialah dengan mengubah nilai biner ke nilai hexadecimal untuk

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

mengetahui suatu karakter yang dihasilkan dari proses kompresi Huffman yang sesuai dengan kode ASCII. Adapun nilai hexadecimal yang sudah terkompresi dapat dilihat pada tabel 7 dibawah ini.

Table 8. Hasil Karakter Terkompresi

Biner	Desimal	Hexadecimal	Karakter
11110011	243	f3	¾
10111100	188	bc	╝
10100000	160	a0	á
10100101	165	a5	Ñ
1011010	90	5a	Z
11000010	194	c2	┬
1001110	78	4e	N
110	6	6	6
1110000	112	70	p
1000001	65	41	A
111	7	7	7

Proses kompresi menghasilkan karakter baru seperti ini "¾╝áÑ Z┬N6pA7". Selanjutnya mengukur hasil kompresi sesuai dengan parameter yang telah ditentukan untuk mengetahui tingkat kinerja algoritma Huffman seperti dibawah ini.

$$Rasio\ of\ Compression\ (RC) = \frac{ukuran\ sebelum\ dikompresi}{ukuran\ setelah\ dikompresi} = \frac{160}{74} = 2.16$$

$$Compression\ of\ Rasio\ (CR) = \frac{ukuran\ sebelum\ dikompresi}{ukuran\ setelah\ dikompresi}\ x\ 100\% = \frac{74}{160}\ x\ 100\% = 46.25\%$$

$$Redundancy\ (Rd) = \frac{sebelum\ dikompresi - setelah\ dikompresi}{ukuran\ sebelum\ dikompresi}\ x\ 100\% = \frac{160 - 74}{160}\ x\ 100\% = 53.75\%$$

$$Space\ Saving\ \ (SS) = 100\% - Compression\ of\ Rasio\ (CR) = 100\% - 46.25\% = 53.75\%$$

Dari perhitungan diatas space saving yang dihasilkan dari kompresi file dokumen dengan menggunakan algoritma Huffman sebesar 53.75%.

3.2 Penerapan Algoritma Levenstein

Algoritma Levenstein merupakan algoritma yang prosesnya melalui tahapan-tahapan tertentu baik pada saat pengkodean maupun pembacaan sandi. Untuk proses enkripsi dalam algoritma Levenstein angka 0 adalah satu 0. Nilai hexadecimal diurutkan berdasarkan frekuensi nilai tertinggi hingga frekuensi nilai terendah kemudian tambahkan kode Levenstein.

Table 9. Nilai *Hexadecimal* Yang Dikompresi Dengan Levenstein

No	Nilai Hexadecimal	Freq	Kode Levenstein	Bit	Freq x Bit
1	31	3	0	1	3
2	5C	3	10	2	6
3	61	2	1100	4	8
4	66	2	1101	4	8
5	30	1	1110 000	7	7
6	38	1	1110 001	7	7
7	6C	1	1110 010	7	7
8	6E	1	1110 011	7	7
9	64	1	1110 1000	8	8
10	65	1	1110 1001	8	8
11	67	1	1110 1010	8	8
12	72	1	1110 1011	8	8
13	74	1	1110 1100	8	8
14	7B	1	1110 1101	8	8
	Total	20	-	-	101

Berdasarkan tabel maka ukuran akhir string bit yang dihasilkan dari kompresi dengan algoritma Levenstein adalah 101 bit. Kemudian sesuaikan kode Levenstein dengan masing-masing nilai hexadecimal.

Tabel 10. Codeword Algoritma Levenstein

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

7B	5C	72	74	66	31	5C	61	64	65
1110 1101	10	1110 1011	1110 1100	1101	0	10	1100	1110 1000	1110 1001
66	**6C**	**61**	**6E**	**67**	**31**	**30**	**38**	**31**	**5C**
1101	1110 010	1100	1110 011	1110 1010	0	1110 000	1110 001	0	10

String yang dihasilkan dari codeword yaitu 101 bit.
"11101101101110101111101100110101011001110100011101001110111100101100111001111101010011100001110001010".
Mencari nilai padding dan flagging

a. Jika sisa bagi panjang string bit terhadap 8 adalah 0 maka tambahkan 00000001. Nyatakan dengan bit akhir.
b. Jika sisa bagi panjang string bit terhadap 8 adalah n (1, 2, 3, 4, 5, 6, 7) maka tambahkan 0 sebanyak 7 – n +"1" di akhir string bit. Lalu tambahkan bilangan biner dari 9 - n. Nyatakan dengan Bit Akhir.

Table 11. Penambahan *Padding* Dan *Flagging*

Padding	*Flagging*
$101\ mod\ 8 = 5 = n$	$9 - n$
$7 - n + "1" = 7 - 5 + "1" = 2 = 001$	$9 - 5 = 4 = 00000100$

Maka tambahkan 001 dan 00000100 ke akhir string bit, menjadi:
"11101101101110101111101100110101011001110100011101001110111100101100111001111101010011100001110001010000100000100".
Setelah ditambah padding dan flagging total bit menjadi 112 bit. Lalu bagi string bit menjadi per 8 bit. Dan ubah menjadi karakter seperti tabel dibawah:

Table 12. Hasil Kompresi Dengan Algoritma Levenstein

Biner	Desimal	Hexadecimal	Karakter
11101101	237	ed	Ý
10111010	186	ba	‖
11111011	251	fb	¹
110101	53	35	5
1100111	103	67	g
1000111	71	47	G
1001110	78	4e	N
11110010	242	f2	
11001110	206	ce	╬
1111101	125	7d	}
1001110	78	4e	N
11100	28	1c	FS
1010001	81	51	Q
100	4	4	4

Setelah file dikompresi dengan algoritma Levenstein, maka menghasilkan karakter baru yaitu "Ý‖¹5gGN_╬}NFSQ". Selanjutnya untuk mengetahui kinerja algoritma Levenstein maka dilakukan perhitungan dengan parameter berikut ini:

$$Rasio\ of\ Compression\ (RC) = \frac{ukuran\ sebelum\ dikompresi}{ukuran\ setelah\ dikompresi} = \frac{160}{101} = 1.58$$

$$Compression\ of\ Rasio\ (CR) = \frac{ukuran\ sebelum\ dikompresi}{ukuran\ setelah\ dikompresi}\ x\ 100\% = \frac{101}{160}\ x\ 100\% = 63.13\%$$

$$Redundancy\ (Rd) = \frac{sebelum\ dikompresi - setelah\ dikompresi}{ukuran\ sebelum\ dikompresi}\ x\ 100\% = \frac{160 - 101}{160}\ x\ 100\% = 36.88\%$$

$$Space\ Saving\ \ (SS) = 100\% - Compression\ of\ Rasio\ (CR) = 100\% - 63.13\% = 36.87\%$$

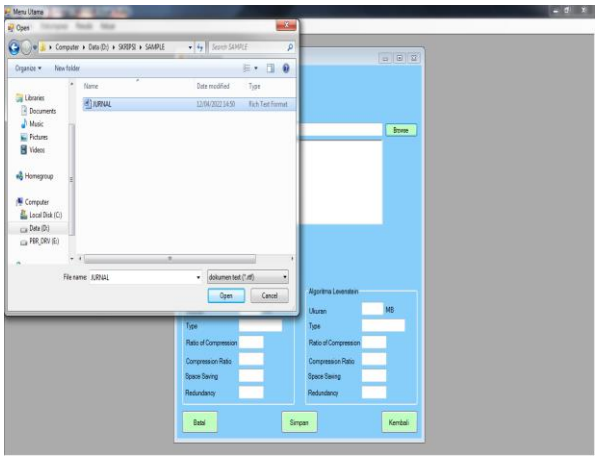
Dari perhitungan diatas space saving yang dihasilkan dari kompresi file dokumen dengan menggunakan algoritma Levenstein sebesar 36.87%.

3.3 Hasil Pengujian

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

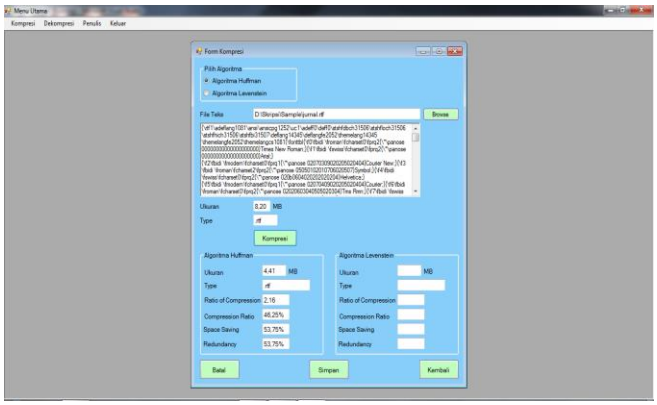
Pengujian sistem adalah langkah yang diambil untuk memastikan bahwa sistem telah berjalan sesuai dengan rancangan yang telah dibuat sebelumnya. Hasil pengujian akan digunakan untuk meningkatkan kinerja sistem dan juga untuk pengembangan sistem lebih lanjut.

Untuk melakukan kompresi setelah memilih tombol kompresi pada menu utama, selanjutnya memilih file yang akan dikompresi dengan mengklik tombol browse maka akan tampil form seperti dibawah ini:



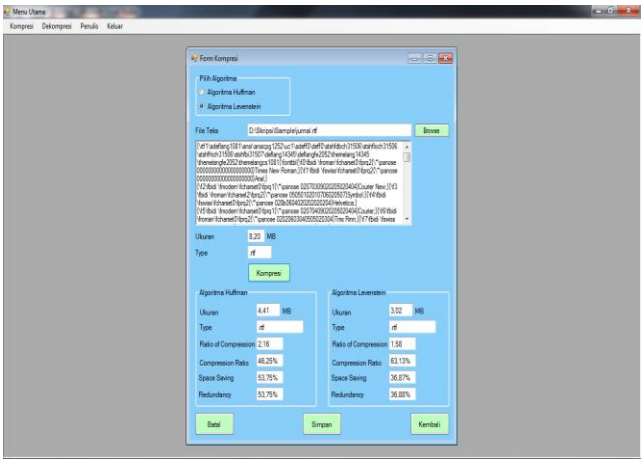
Gambar 15. Tampilan *Browse* Pada *Form* Kompresi

Setelah memilih file untuk dikompresi, selanjutnya melakukan kompresi dengan algoritma Huffman dan sistem akan menampilkan hasil perhitungan seperti gambar dibawah ini:



Gambar 16. Tampilan Kompresi Dengan Algoritma Huffman

Kemudian lakukan kompresi dengan algoritma Levenstein dan terlihat hasil kompresi seperti dibawah ini.



Gambar 17. Tampilan Kompresi Dengan Algoritma Levenstein

Setelah file berhasil dikompresi dengan kedua algoritma maka dapat terlihat hasil kompresi algoritma mana yang lebih kecil. Kemudian simpan file hasil kompresi dengan algoritma yang diinginkan.

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

Dari pengujian yang dilakukan maka menghasilkan perbedaan ukuran file yang telah dikompresi dengan algoritma Huffman dan algoritma Levenstein terlihat pada tabel dibawah ini:

Table 13. Hasil Pengujian Algoritma Huffman

No	Nama File	Ukuran Sebelum Dikompresi	Ukuran Setelah Dikompresi	Algoritma Levenstein			
				RC	CT (%)	RD (%)	SS (%)
1	Jurnal.RTF	8.2 MB	3.79 MB	1.58	63.13	36.88	36.88
2	Skripsi.RTF	49.6 MB	15.81 MB	2,71	36,88	63,13	63,12
3	Test Data.RTF	5 MB	2.19 MB	1,63	61,25	38,75	38,75

Table 14. Hasil Pengujian Algoritma Levenstein

No	Nama File	Ukuran Sebelum Dikompresi	Ukuran Setelah Dikompresi	Algoritma Levenstein			
				RC	CT (%)	RD (%)	SS (%)
1	Jurnal.RTF	8.2 MB	5.18 MB	1.58	63.13	36.88	36.88
2	Skripsi.RTF	49.6 MB	18.29 MB	2,71	36,88	63,13	63,12
3	Test Data.RTF	5 MB	3.06 MB	1,63	61,25	38,75	38,75

Berdasarkan tabel diatas, maka yang menjadi algoritma terbaik dalam kompresi file dokumen yaitu algoritma Huffman. Dimana algoritma Huffman memiliki space saving yang lebih besar dan menghasilkan ukuran hasil kompresi yang lebih kecil dibandingkan dengan algoritma Levenstein.

4. KESIMPULAN

Berdasarkan hasil implementasi dan pengujian yang dilakukan terhadap penelitian analisis perbandingan kinerja algoritma Huffman dan algoritma Levenstein dalam kompresi file dokumen, maka dapat ditarik kesimpulan bahwa, prosedur kompresi file dokumen dengan menggunakan algoritma Huffman dan algoritma Levenstein berhasil dilakukan dengan file dokumen format .RTF dan proses kompresi berjalan sesuai dengan teknik kompresi. Penerapan algoritma Huffman dan algoritma Levenstein telah diterapkan dan terbukti dapat mengkompresi file dokumen format .RTF. Hasil pengujian file dokumen format .RTF dengan algoritma Huffman dan algoritma Levenstein memperlihatkan bahwa algoritma Huffman lebih baik dalam melakukan kompresi file dokumen. Dimana ukuran hasil kompresi lebih rendah dan space saving yang dihasilkan lebih tinggi daripada algoritma Levenstein. Penelitian ini berhasil membangun sebuah aplikasi yang dapat melakukan proses kompresi dan dekompresi file dokumen dengan menerapkan algoritma Huffman dan algoritma Levenstein menggunakan aplikasi Microsoft Visual Studio 2008.

REFERENCES

[1] N. Aftikasyah, “Penerapan Algoritma Yamamoto ’ s Recursive Code Untuk Mengkompresi File Dokumen,” vol. 5, pp. 255–263, 2022, doi: 10.30865/komik.v5i1.3716.

[2] E. Prayoga and K. M. Suryaningrum, “Implementasi Algoritma Huffman Dan Run Length Encoding Pada Aplikasi Kompresi Berbasis Web,” J. Ilm. Teknol. Infomasi Terap., vol. 4, no. 2, pp. 92–101, 2018, doi: 10.33197/jitter.vol4.iss2.2018.154.

[3] L. V. Simanjuntak, “Perbandingan Algoritma Elias Delta Code dengan Levenstein Untuk Kompresi File Teks,” vol. 1, no. 3, pp. 184–190, 2020.

[4] H. Sinaga, P. Sihombing, and H. Handrizal, “Perbandingan Algoritma Huffman Dan Run Length Encoding Untuk Kompresi File Audio,” Talent. Conf. Ser. Sci. Technol., vol. 1, no. 1, pp. 010–015, 2018, doi: 10.32734/st.v1i1.183.

[5] N. Sari, “Perancangan Aplikasi kamus Bahasa Minang Indonesia Dan Indonesia Minang Menggunakan Algoritma Levenshtein,” J. Mhs. Fak. Tek. dan Ilmu Komput., vol. 1, no. 1, pp. 1113–1124, 2020, [Online]. Available: http://e-journal.potensi-utama.ac.id/ojs/index.php/FTIK/article/view/950.

[6] Supriyadi and O. Frida, “Analisis Perbandingan Pemampatan Data Teks Dengan Menggunakan Metode Huffman Dan Half – Byte,” Algoritm. J. Ilmu Komput. dan Inform., vol. 2, no. 1, pp. 1–6, 2018.

[7] D. Pratiwi and T. Zebua, “Analisis Perbandingan Kinerja Algoritma Fixed Length Binary Encoding Dan Algoritma Elias Gamma Code Dalam Kompresi File Teks,” KOMIK (Konferensi Nas. Teknol. Inf. dan Komputer), vol. 3, no. 1, pp. 424–430, 2019, doi: 10.30865/komik.v3i1.1623.

[8] A. Pahdi, “Algoritma Huffman Dalam Pemampatan Dan Enksripsi Data,” IJNS - Indones. J. Netw. Secur., vol. 6, no. 3, pp. 1–7, 2017, [Online]. Available: http://ijns.org/journal/index.php/ijns/article/view/1461.

[9] N. F. Rizky, S. D. Nasution, and F. Fadlina, “Penerapan Algoritma Elias Delta Codes Dalam Kompresi File Teks,” Build. Informatics, Technol. Sci., vol. 2, no. 2, pp. 109–114, 2020, doi: 10.47065/bits.v2i2.138.

[10] B. A. ANASTA, “ANALISIS PERBANDINGAN ALGORITMA RUN-LENGTH ENCODING DAN ALGORITMA FIBONACCI CODE DALAM KOMPRESI CITRA,” vol. 1, no. 3, pp. 82–91, 2018.

[11] R. Yanur Tanjung, “Perancangan Aplikasi Kompresi File Dokumen Menggunakan Algoritma Adiitive Code,” J. Ris. Komputer), vol. 8, no. 4, pp. 2407–389, 2021, doi: 10.30865/jurikom.v8i4.3593.

ISSN 2597-4645 (media online)
ISSN 2597-4610 (media cetak)
Page: 87-99

[12] R. D. A. RAWIE, "PERBANDINGAN KINERJA ALGORITMA ELIAS OMEGA CODE DAN ALGORITMA LEVENSTEIN CODE DALAM KOMPRESI CITRA," p. 6, 2021.